Voraussetzungen

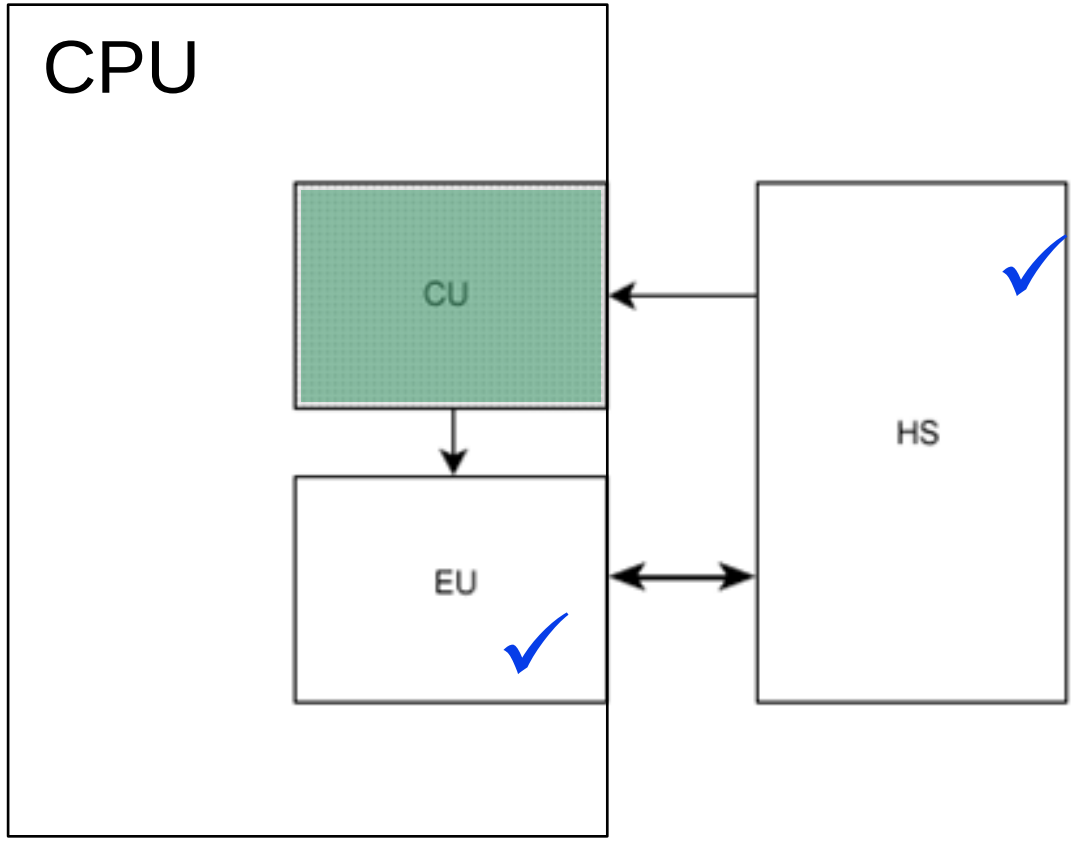
- von-Neumann-Rechner
- Kombinatorische und sequenzielle Logik

Ziel

- Detailliertes Verständnis und Anwendung von Steuermechanismen

Inhalt

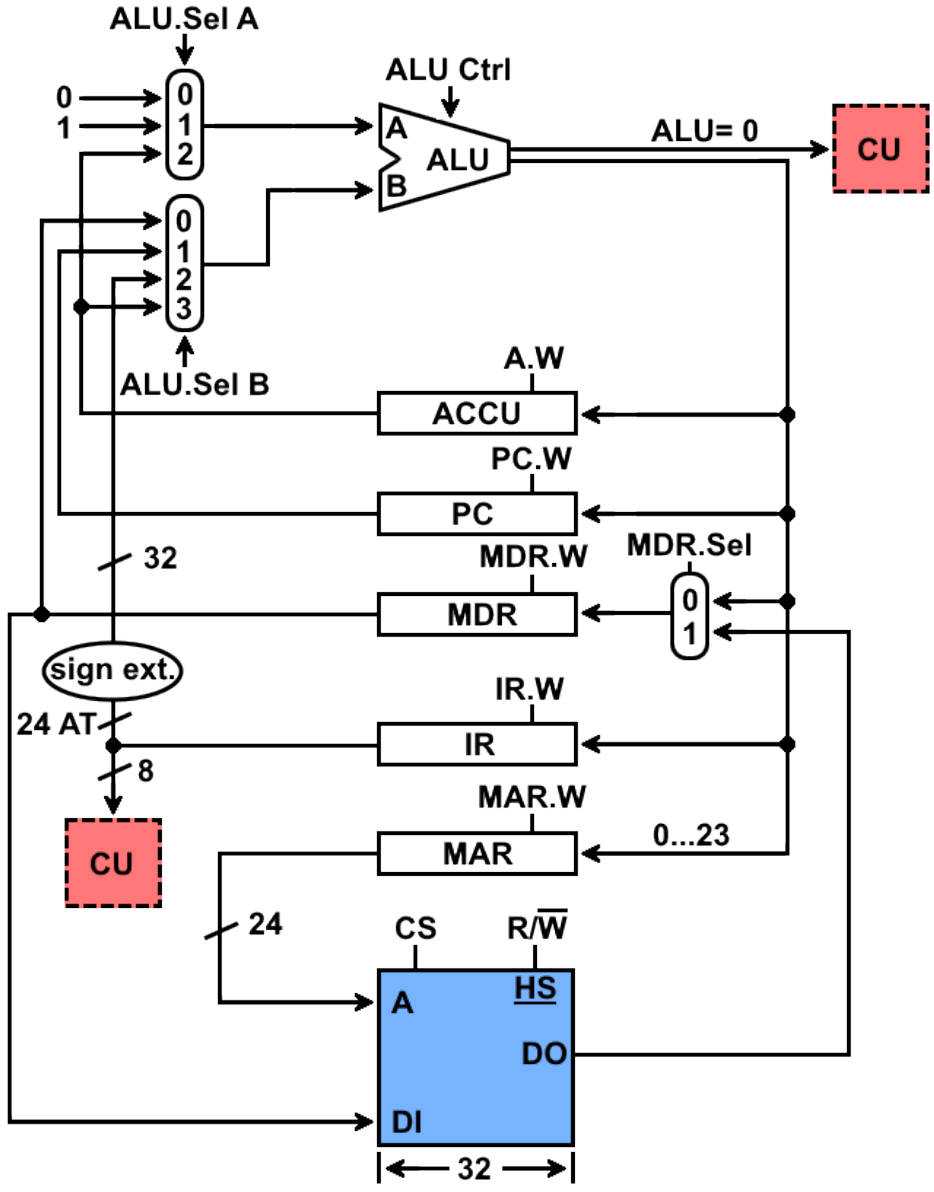
- MINIMAX-Beispielmaschine
- CU-Aufgaben
- Interpretation von Maschinenbefehlen
- Steuersignale
- Mikrobefehle und -operationen
- CU nach Wilkes
- Endlicher Automat als CU
- Sequenzzähler als CU
- Mikroprogrammierung
- Mikroprogrammierte CU



RA - T11 - 2

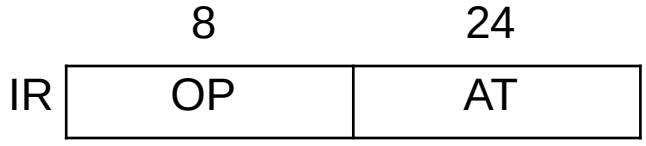
- Zum genaueren Verständnis des Datenpfadaufbaus und anschließend zur Entwicklung der Steuerung wird die einfache Ein-Adress-Beispielmaschine MINIMAX verwendet.

- Sie hat folgende Eigenschaften:
  - 1 ALU für arithmetische Operationen, auch zur Adressrechnung verwendet
  - 1 bit-Flagregister: COND = 1 wenn ALU == 0
  - arithmetische Operationen verknüpfen ACCU und MDR (MDR: Memory Data Register)
  - ALU-Eingänge A und B werden belegt über MUXe; keine Busse
  - HS: Adresse 24 bit, Daten 32 bit, wortadressiert (32-bit)



MDR: Memory Data Register
MAR: Memory Address Register
HS.DO: Hauptspeicher.Data out
HS.DI: Hauptspeicher.Data in

- ALU und Register (bis auf MAR: Memory Address Register): 32 bit
  MAR: 24 bit
- Befehlsformat:



Opcode OP: 8 bit
Adressteil AT: 24 bit

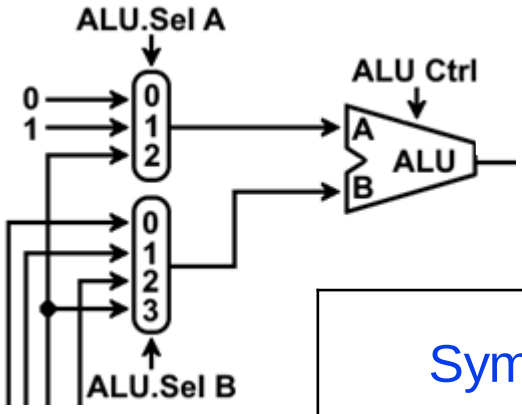
- Die Register sind als zweiflankengesteuerte MS-FFs ausgelegt. Damit kann ein Register während eines Taktes zunächst als Quelle und dann als Ziel dienen.
- Der Adressteil AT sowie der Opcode OP werden aus IR (IR: Instruktions-register) gewonnen:

$$AT_{23...0} \leftarrow IR_{23...0};$$
$$OP_{7...0} \leftarrow IR_{31...24};$$

- AT wird durch die Sign Extension Unit vorzeichenrichtig auf 32 bit aufgefüllt.

☐ Wie groß ist der adressierbare Speicher der MINIMAX Maschine maximal?

- 1 KiByte
- 16 KiByte
- 1MiByte
- 16 MiByte
- 64 MiByte
- 1 GiByte

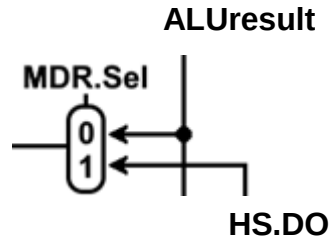


ALU kennt 4 Operationen

Symbol	ALU-Operation	ALU Ctrl
ADD	ALUresult ← A + B	00
SUB	ALUresult ← -A + B	01
TRANS.A	ALUresult ← A	10
TRANS.B	ALUresult ← B	11

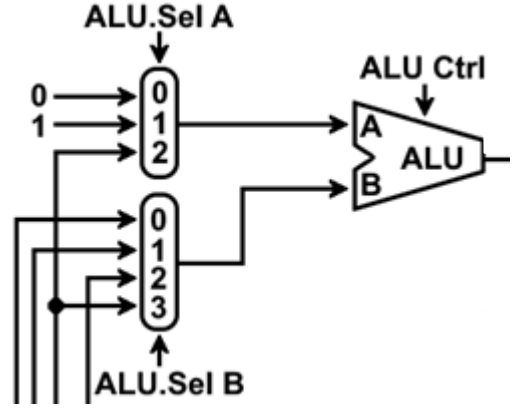
ALUSel.A	A
0	0
1	1
2	ACCU

MDR.Sel	MDR
0	ALUresult
1	HS.DO

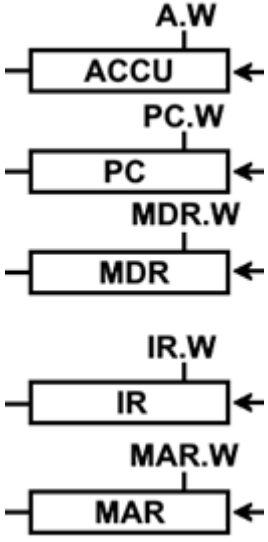


HS.DO: Hauptspeicher.Data out

ALUSel.B	B
0	MDR
1	PC
2	$(IR_{23})^8@IR_{23..0}$
3	ACCU



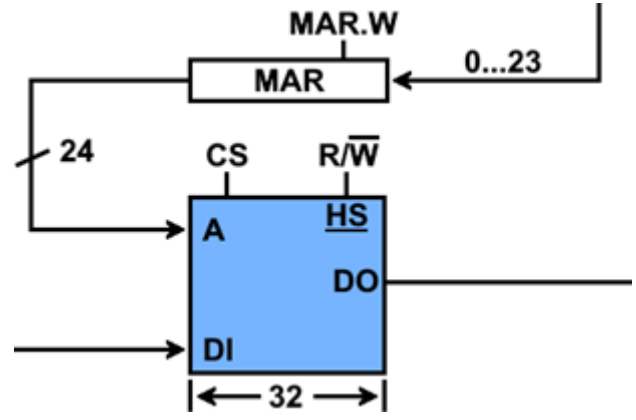
ALUSel.B=2: Auffüllen mit Wert von Bit 23 mittels sign. ext.

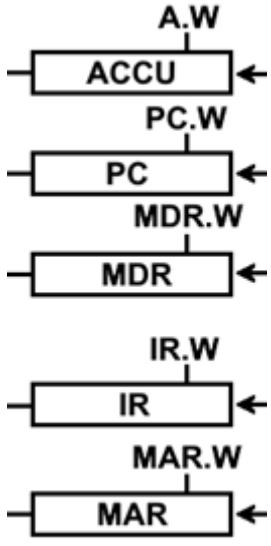


Schreibsignale A.W, PC.W, MDR.W, IR.W, MAR.W	Operation
0	-
1	schreiben

HS		Operation
CS	R/$\overline{W}$	
0	0	-
0	1	-
1	0	M(MAR) ← HS.DI
1	1	HS.DO ← M(MAR)

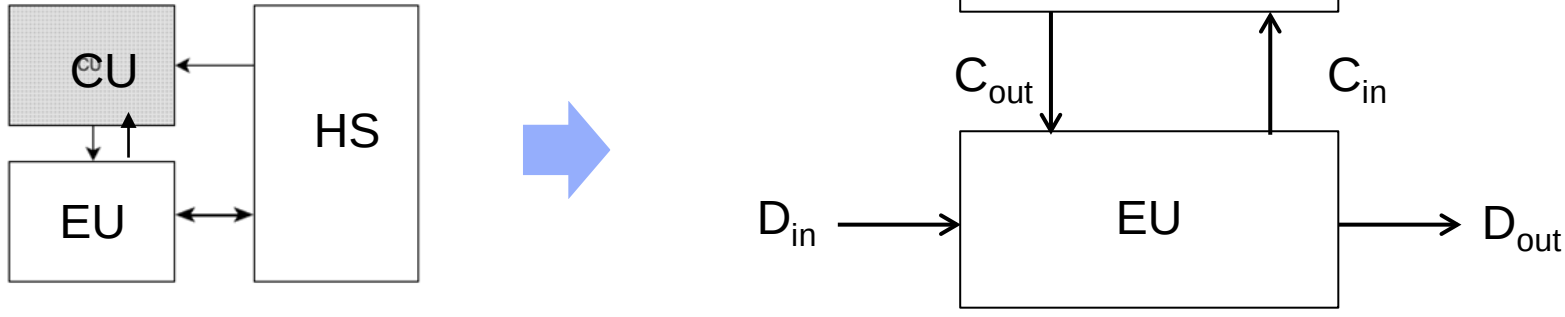
HS.DO: Hauptspeicher.Data out
HS.DI: Hauptspeicher.Data in





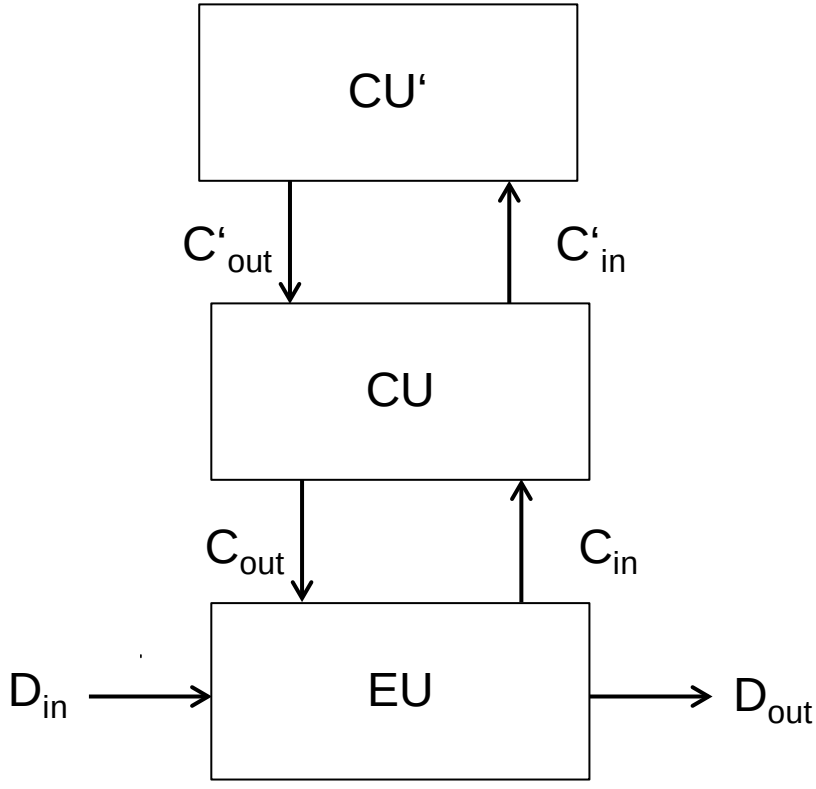
Register	W	
ACCU	32	Accumulator
PC	32	Program Counter; $PC_{31..24} = 0$ (Speicheradresse 24 bit breit)
MDR	32	Memory Data Register
IR	32	Instruktionsregister
MAR	24	Memory Address Register

- Die *Ausführungseinheit* (EU) kann dargestellt werden als eine Black Box, welche
  - (1) einen Dateneingang D_{in} und einen Datenausgang D_{out} besitzt (+ Adressleitungen)
  - (2) durch Leitungen C_{out} gesteuert wird,
  - (3) Signale C_{in} an die Steuerung zurückgibt.



- Aufgabe der Steuereinheit (Control Unit, CU) ist es, die Steuerleitungen C_{out} in der richtigen Abfolge zu aktivieren und dabei die von der EU erhaltenen Signale C_{in} zu berücksichtigen.

- Häufig werden Steuerungen geschachtelt eingesetzt, so dass eine Hierarchie entsteht.



- Ein typischer Mikroprozessor benötigt etwa 100 bis 150 Steuerleitungen c_i^{out}.
- Jede Steuerleitung ist dabei (u.U. zusammen mit anderen) für die Ausführung einer elementaren RT-Operation verantwortlich.
- Die CU greift auf drei Arten steuernd in die EU ein:
  - Sie führt den Datentransfer zwischen Registern durch, wobei auch zwischen unterschiedlichen Quellen unterschieden werden muss (Datenlenkung).
  - Sie wählt eine von mehreren möglichen Operationen aus (Operationsselektion).
  - Sie gibt ein Taktschema vor.

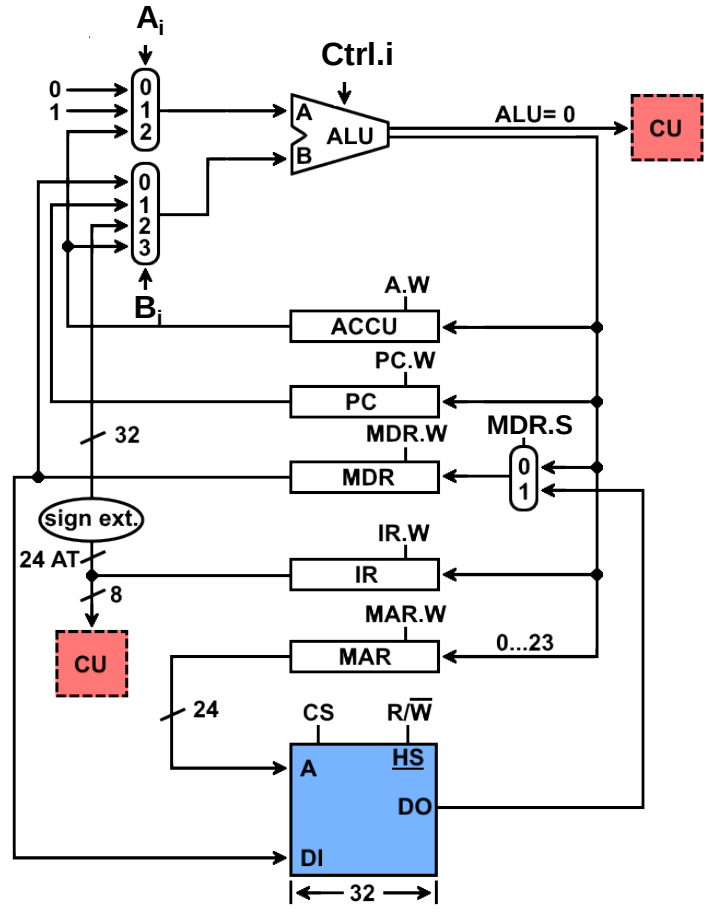
- Steuerungen können festverdrahtet (hardwired control) oder durch Mikroprogrammierung realisiert werden.
- In beiden Fällen sind die Maschinenbefehle in der Form von Sequenzen von RT-Operationen darzustellen.
- Es soll zunächst dieser Übersetzungsvorgang Maschinenbefehl → RT-Operation an Hand der Beispielmaschine MINIMAX dargestellt werden.
- Die Umsetzung jeweils eines Maschinenbefehls in eine Folge von RT-Operationen stellt die oberste HW-Interpretationsebene dar.
- Im Folgenden werden für MINIMAX einige Maschinenbefehle in RT-Operationen übersetzt.

- Geben Sie die Steuersignale für den folgenden Mikrobefehl an:

ACCU ← ACCU + 1 : MDR ← M[MAR]

- Steuersignaltabelle:

A_0	A_1	B_0	B_1	MDR.S	CS	R/$\overline{W}$
Ctrl.0	**Ctrl.1**	**A.W**	**PC.W**	**MDR.W**	**IR.W**	**MAR.W**



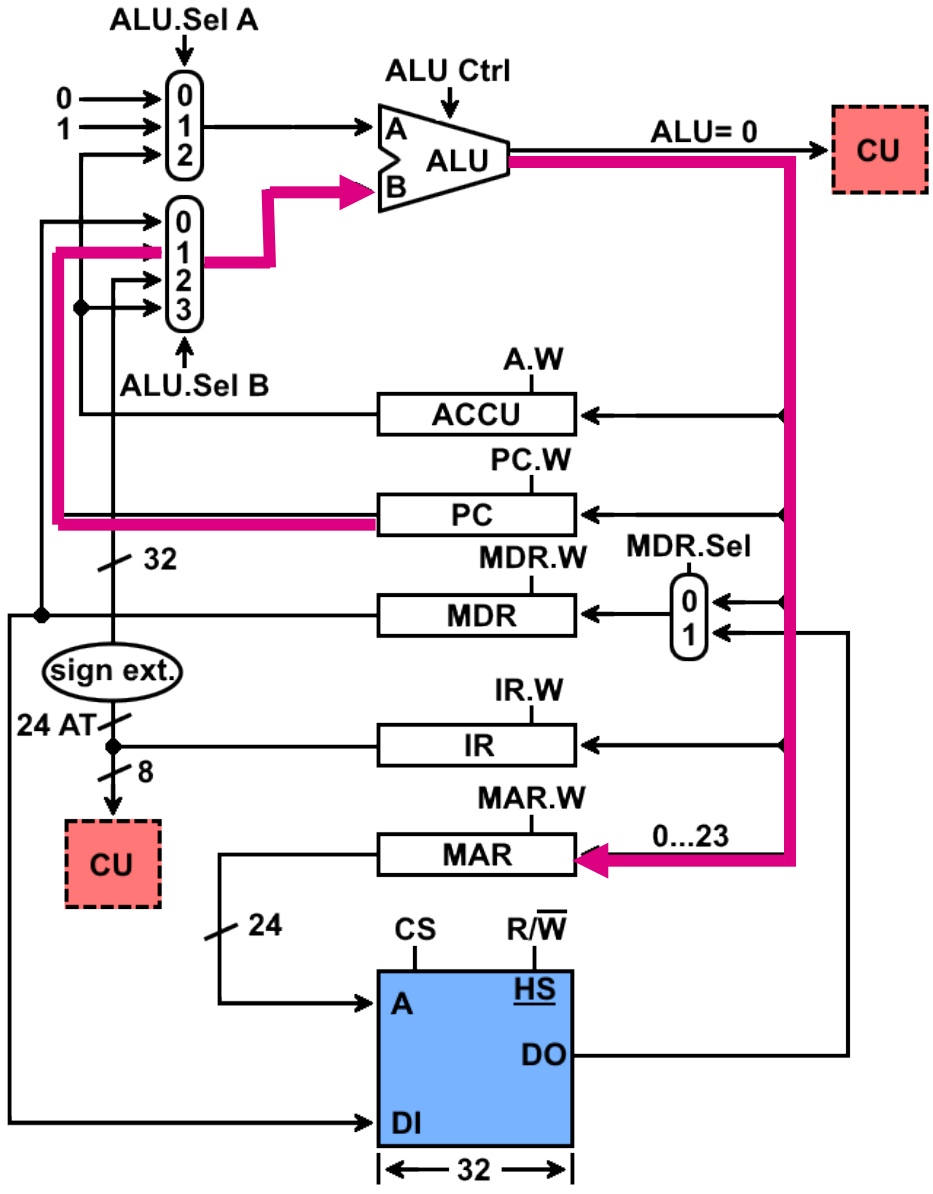
- Allen Befehlen gemeinsam ist die IFETCH-Phase.

- IFETCH:

```
MAR ← PC;          /* Befehlszähler adressiert Befehl im Speicher
MDR ← M [MAR];     /* HS read
IR  ← MDR;         /* CU ← OP
PC  ← PC + 1;      /* Befehle sind i.d.R. aufeinander folgend
GOTO OP;           /* CU-interner Befehl, s.u.
```

- Kommentar zu PC ← PC + 1 :
  Inkrement zum nächsten Befehl = 1 bei wortadressiertem Speicher,
  ansonsten ein Vielfaches von Byte, z.B. PC ← PC + 4 bei 32-Bit-Architektur

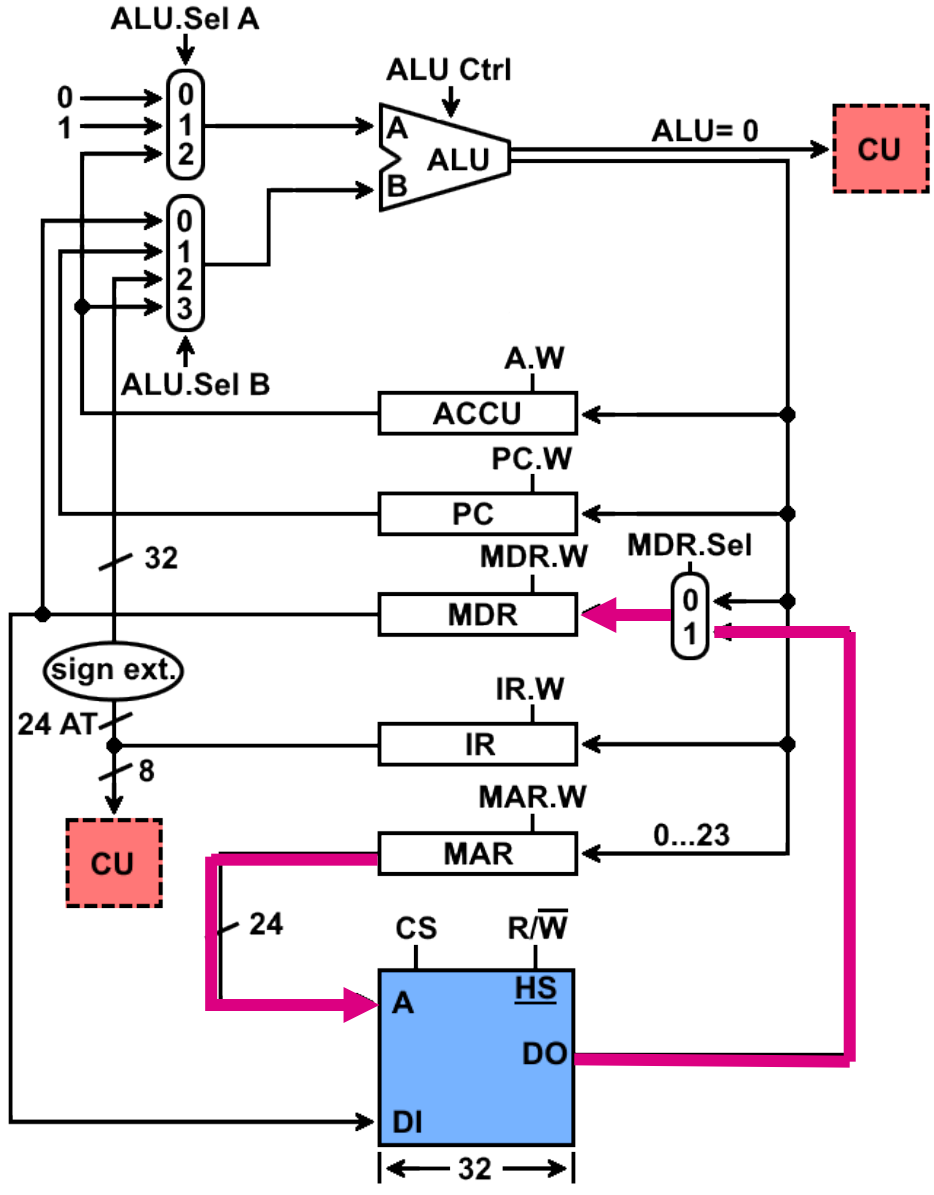
ALU Ctrl	1	1
ALU.Sel B	0	1
MAR.W		1



IFETCH:
MAR ← PC;
MDR ← M [MAR];
IR ← MDR;
PC ← PC + 1;
GOTO OP;

MDR: Memory Data Register
MAR: Memory Address Register
HS.DO: Hauptspeicher.Data out
HS.DI: Hauptspeicher.Data in

CS	1
R/$\overline{W}$	1
MDR.Sel	1
MDR.W	1



IFETCH:

MAR ← PC;

MDR ← M [MAR];

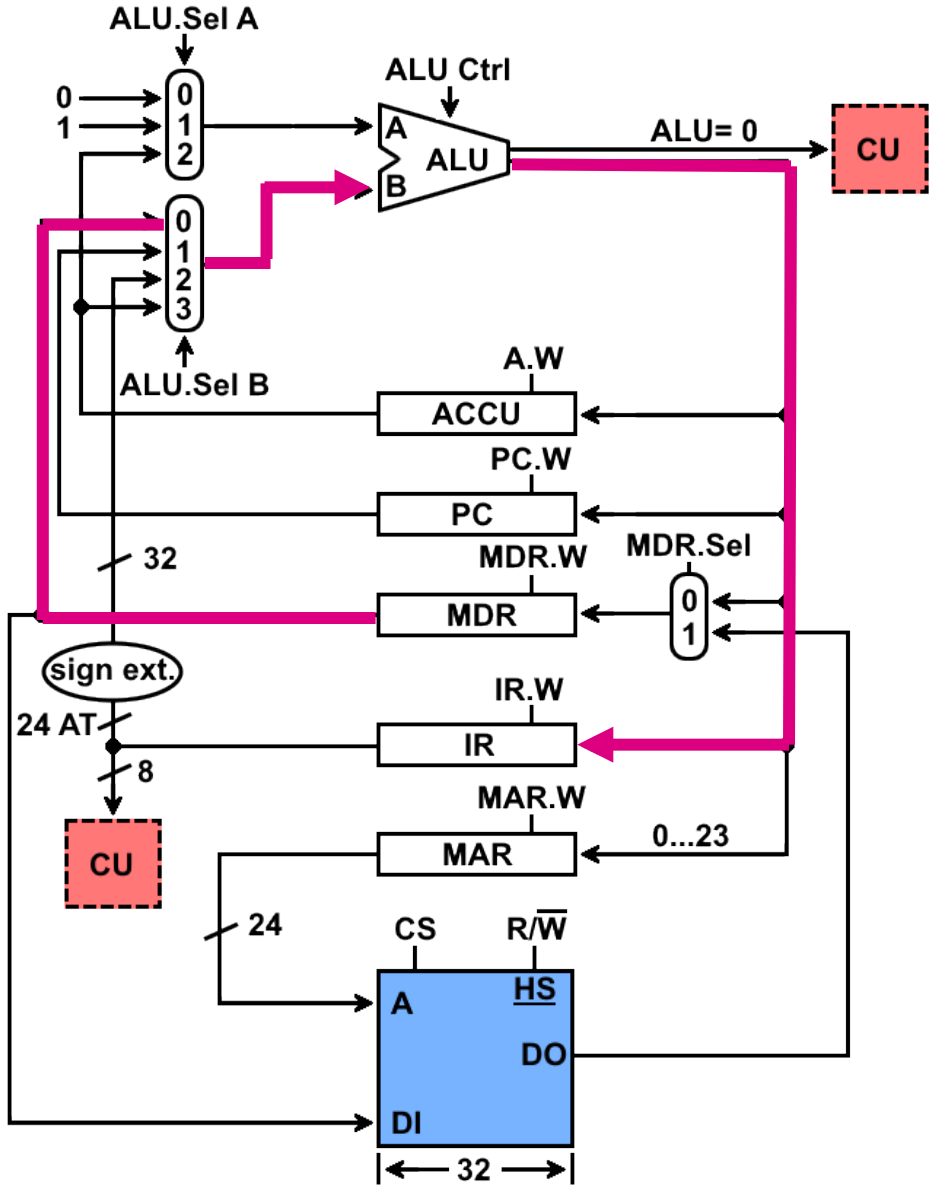
IR ← MDR;

PC ← PC + 1;

GOTO OP;

MDR: Memory Data Register
MAR: Memory Address Register
HS.DO: Hauptspeicher.Data out
HS.DI: Hauptspeicher.Data in

ALU Ctrl	1	1
ALU.Sel B	0	0
IR.W		1



IFETCH:

MAR ← PC;

MDR ← M [MAR];

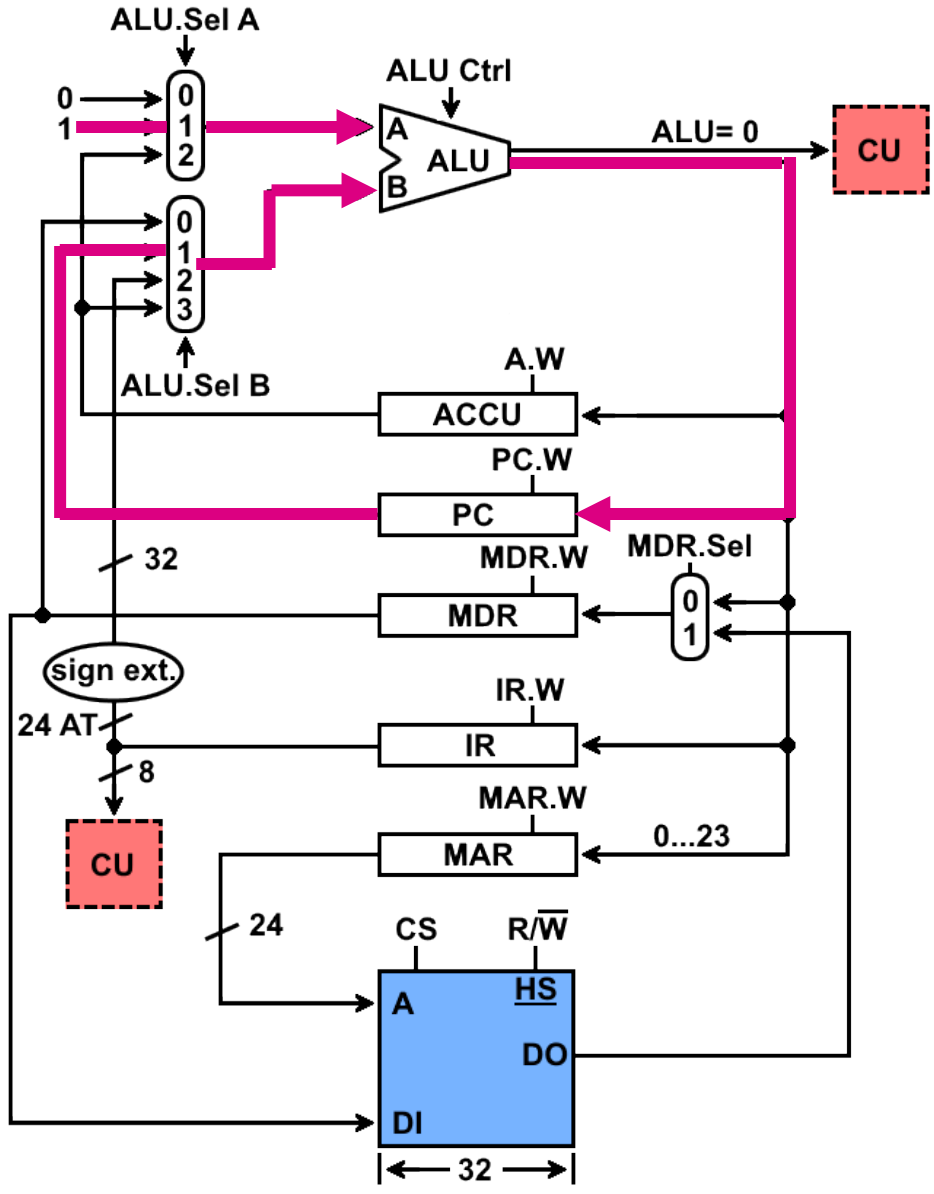
IR ← MDR;

PC ← PC + 1;

GOTO OP;

MDR: Memory Data Register
MAR: Memory Address Register
HS.DO: Hauptspeicher.Data out
HS.DI: Hauptspeicher.Data in

ALU Ctrl	0	0
ALU.Sel A	0	1
ALU.Sel B	0	1
A.W		1



IFETCH:

MAR ← PC;

MDR ← M [MAR];

IR ← MDR;

PC ← PC + 1;

GOTO OP;

MDR: Memory Data Register
MAR: Memory Address Register
HS.DO: Hauptspeicher.Data out
HS.DI: Hauptspeicher.Data in

- LOAD /* (ACCU ← M[AT]):

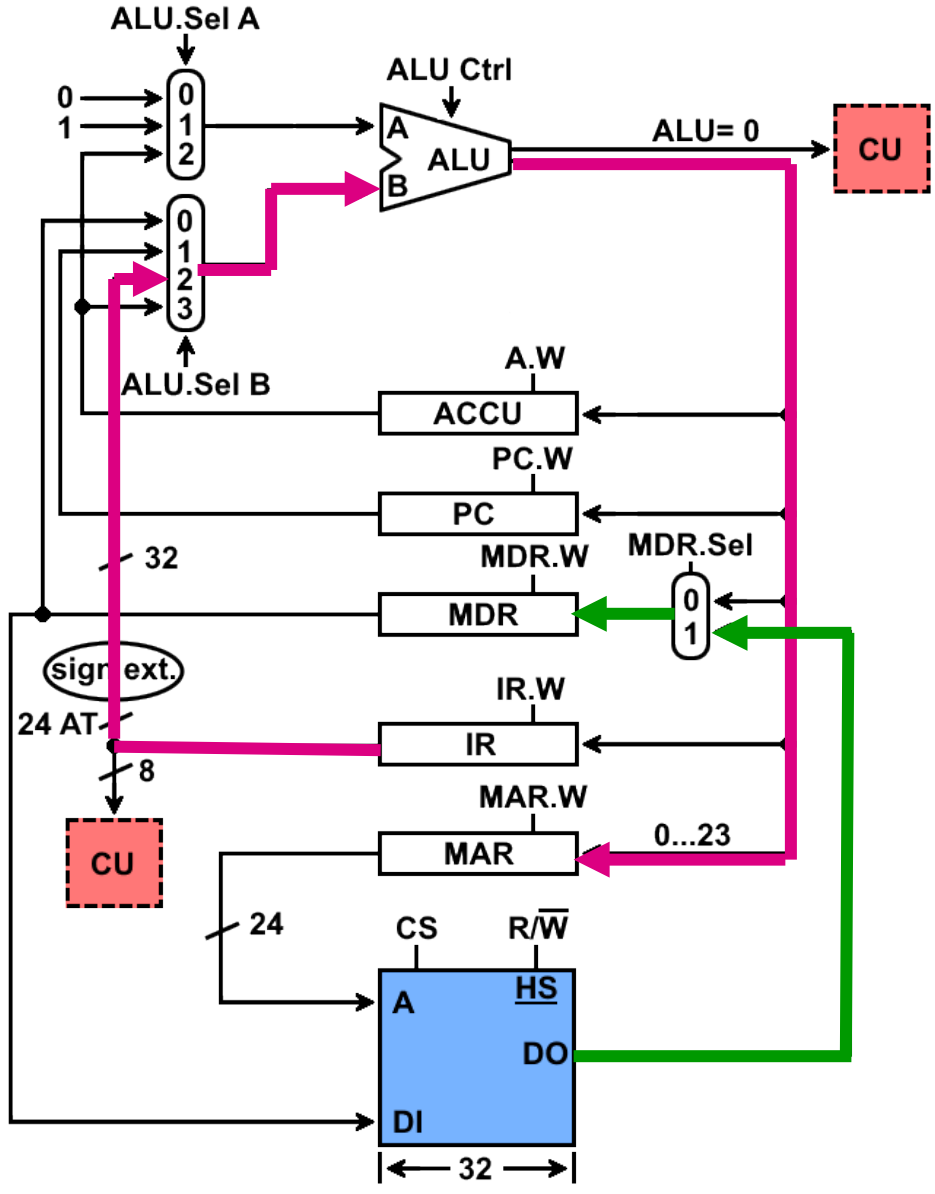
```
MAR     ← AT;          /* Adresse aus Adressteil des Befehls
MDR     ← M [MAR];     /* HS read
ACCU    ← MDR;
GOTO IFETCH;           /* Nächster Befehl
```

- STORE /* (M[AT] ← ACCU):

```
MAR     ← AT;          /* Adresse aus Adressteil des Befehls
MDR     ← ACCU;
M [MAR] ← MDR;         /* HS write
GOTO IFETCH;           /* Nächster Befehl
```

ALU Ctrl	1 1
ALU.Sel B	1 0
MAR.W	1

CS	1
R/$\overline{W}$	1
MDR.Sel	1
MDR.W	1



LOAD

MAR ← AT;
MDR ← M [MAR];
ACCU ← MDR;
GOTO IFETCH;

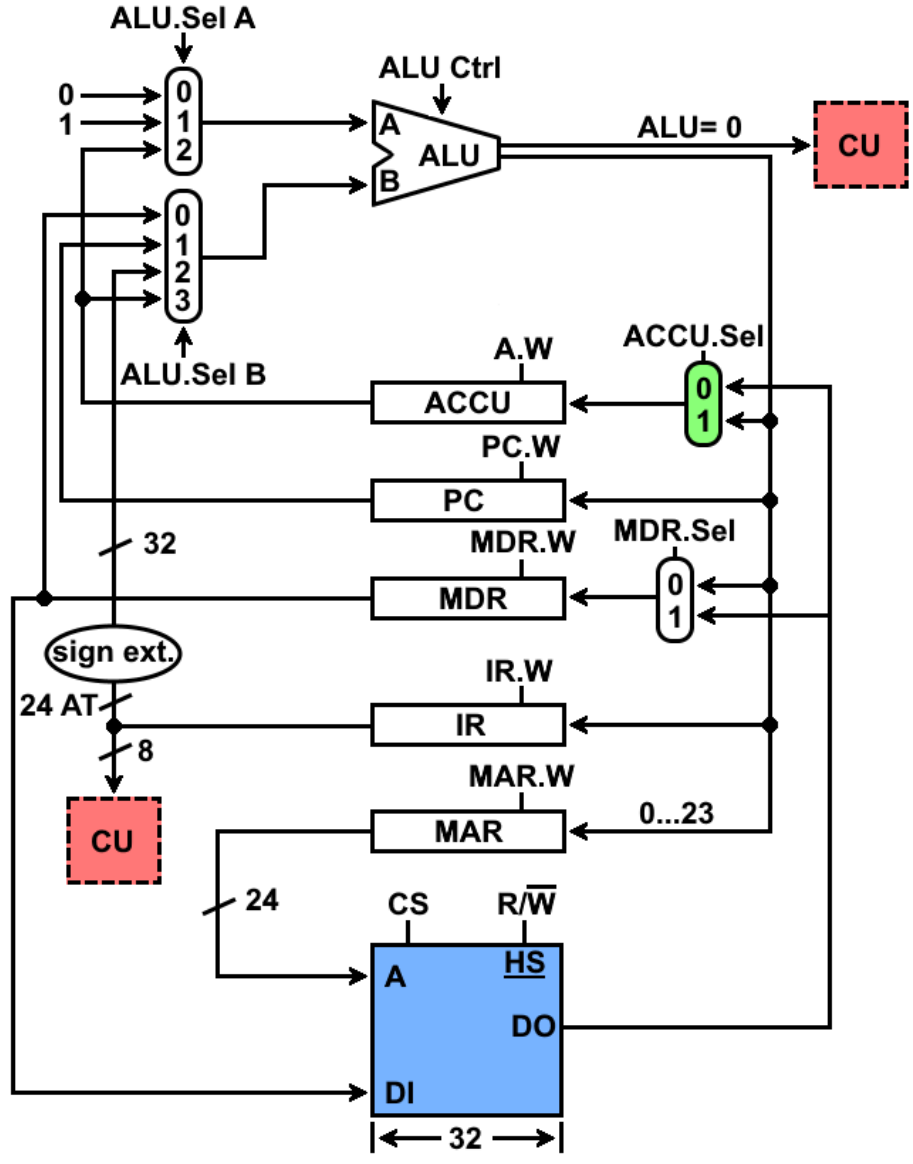
STORE

MAR ← AT;
MDR ← ACCU
M [MAR] ← MDR;
GOTO IFETCH;

MDR: Memory Data Register
MAR: Memory Address Register
HS.DO: Hauptspeicher.Data out
HS.DI: Hauptspeicher.Data in

- Ein LOAD Befehl benötigt in der MINIMAX Maschine 4 Operationen um umgesetzt zu werden.

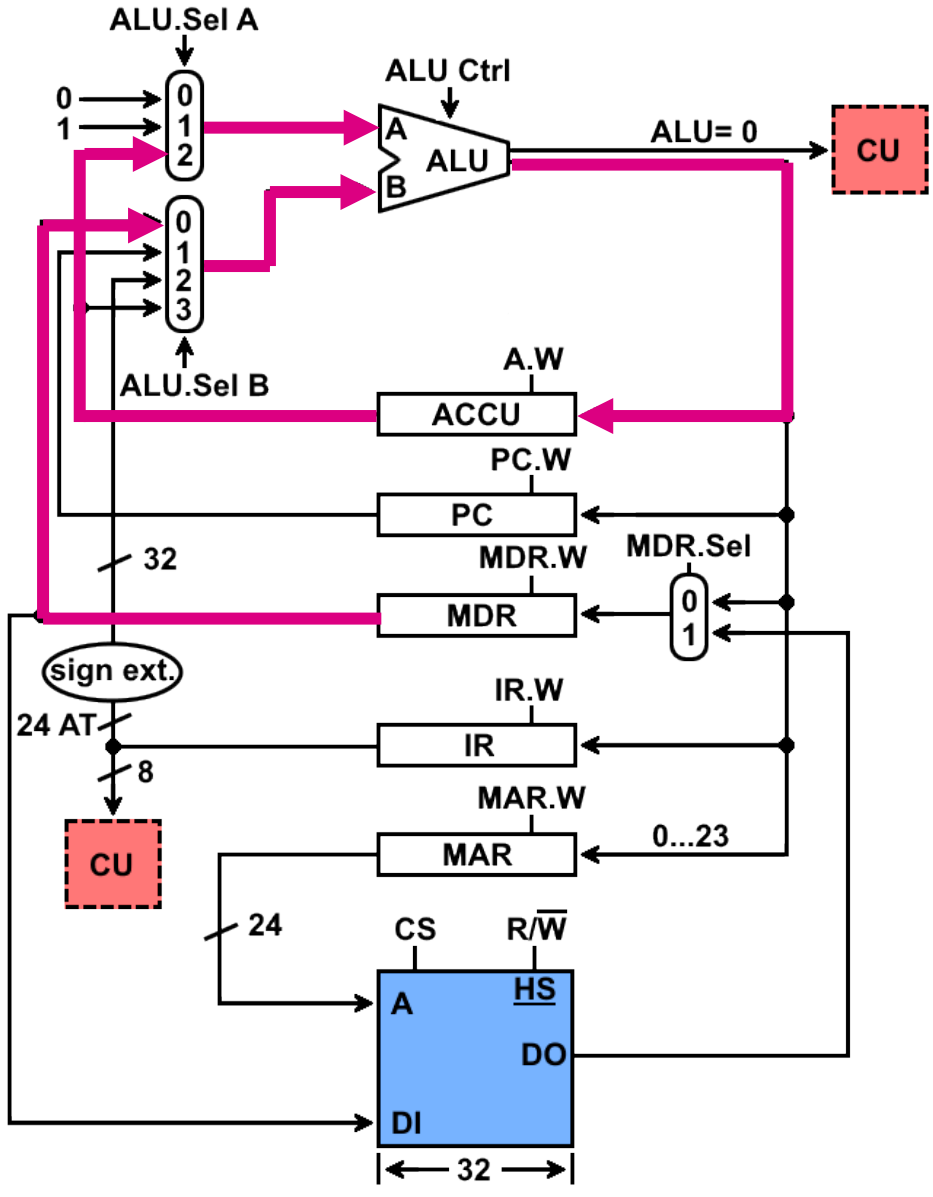
- Wenn nun ein weiterer Multiplexer eingebaut wird (in der Grafik grün markiert), wie viele Operationen umfasst der neue LOAD Befehl?



- ADD /* (ACCU ← ACCU + M[AT]):

```
MAR ← AT;              /* Adresse aus Adressteil des Befehls
MDR ← M[MAR];          /* HS read
ACCU ← ACCU + MDR;     /* ACCU ist mit 1. Operanden bereits geladen
GOTO IFETCH;           /* Nächster Befehl
```

ALU Ctrl	0	0
ALU.Sel A	1	0
ALU.Sel B	0	0
A.W		1



ADD

MAR ← AT;

MDR ← M[MAR];

ACCU ← ACCU + MDR;

GOTO IFETCH;

MDR: Memory Data Register
MAR: Memory Address Register
HS.DO: Hauptspeicher.Data out
HS.DI: Hauptspeicher.Data in

- JUMP (unbedingter Sprung nach AT):

  PC ← AT; /* Adressteil des Befehls adressiert nächsten Befehl

  GOTO IFETCH; /* Nächster Befehl

- BZ (Branch Zero, springe nach AT, falls COND == 1):

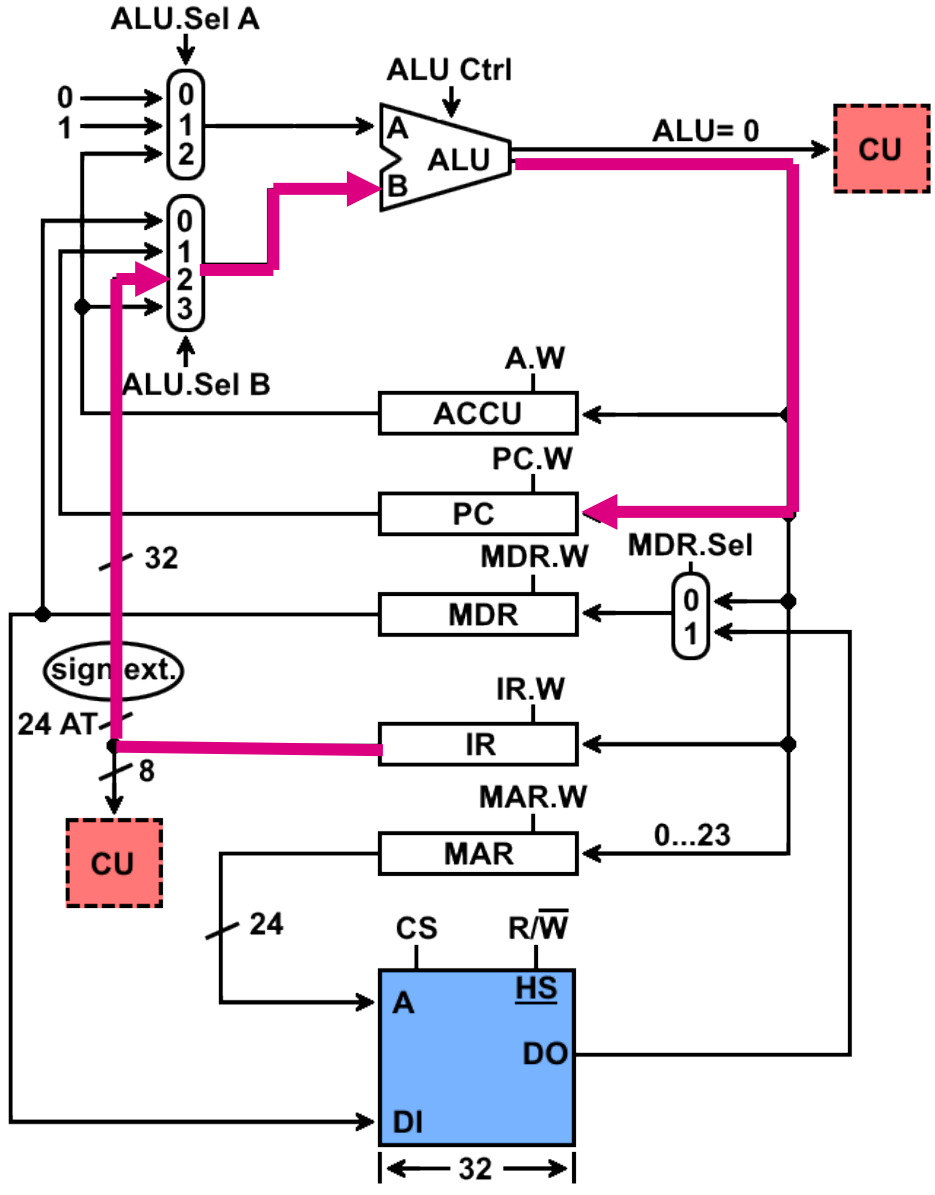
  IF (COND == 1) PC ← AT; /* COND=1, wenn ALU-Ausgang=0

  GOTO IFETCH; /* Nächster Befehl

- CALL UP (Unterprogrammaufruf):

  Unterprogrammaufruf bei der Minimax nicht möglich, es fehlen Register zum Retten des PC und/oder Register für Stackpointer

ALU Ctrl	1	1
ALU.Sel B	1	0
PC.W		1



JUMP

PC ← AT;

GOTO IFETCH;

BZ

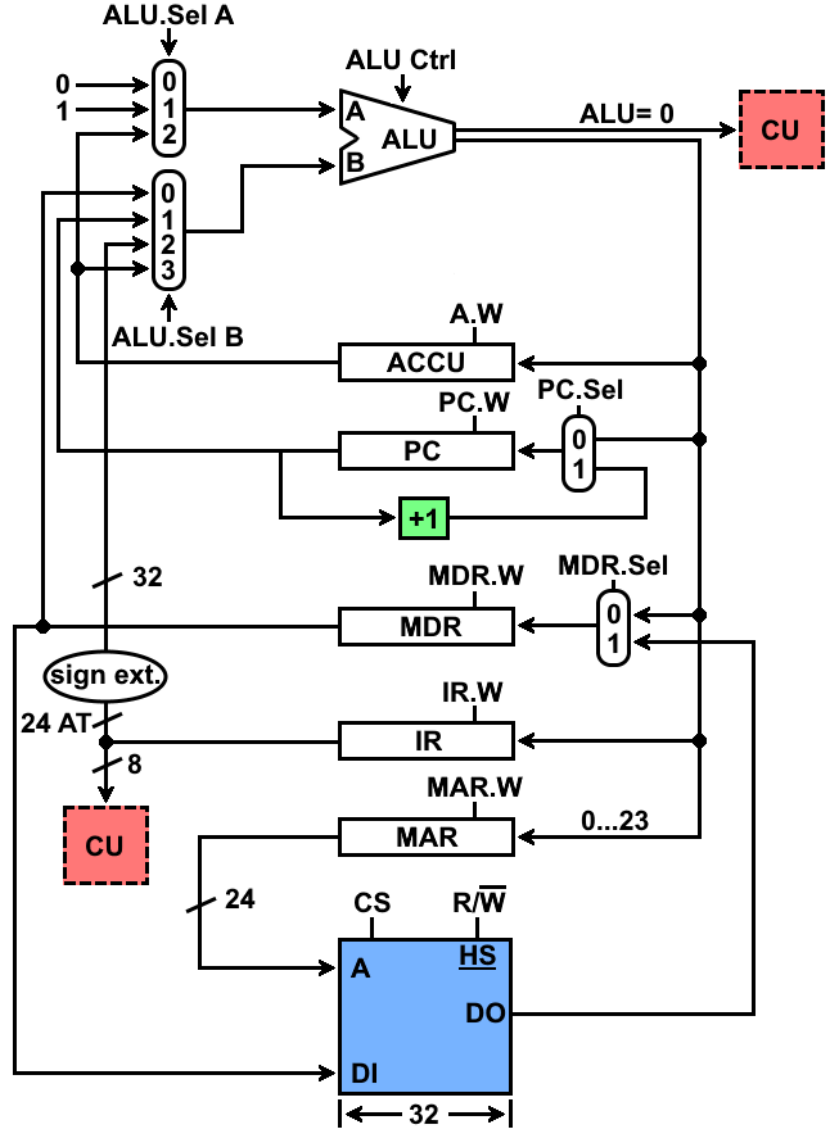
IF (COND == 1) PC ← AT;

GOTO IFETCH;

MDR: Memory Data Register
MAR: Memory Address Register
HS.DO: Hauptspeicher.Data out
HS.DI: Hauptspeicher.Data in

- Ein IFETCH Befehl benötigt in der MINIMAX Maschine 5 Operationsschritte um umgesetzt zu werden.

- Wenn nun ein Inkrementer eingebaut wird (in der Grafik grün markiert), wie viele Operationsschritte ließen sich einsparen?



- Jede RT-Operation wird aktiviert durch Setzen bestimmter Steuersignale. Die folgende Tabelle zeigt die Steuersignale c_0 ... c_{14} für die Maschinenbefehle ADD, JUMP und BZ.

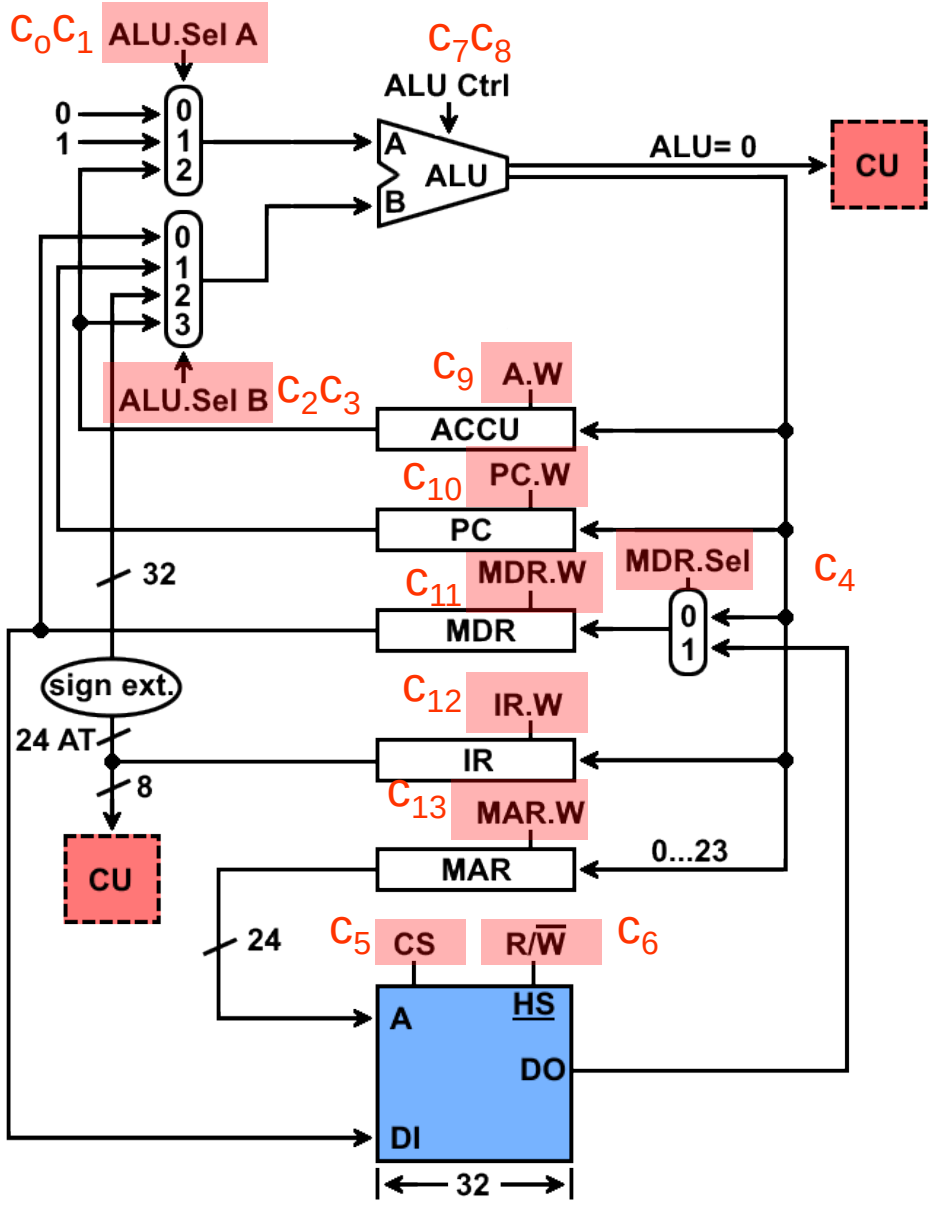
- Die RT-Operationssequenz IFETCH wird jedem Befehl vorangestellt.

- c_0 ... c_{14} sind Ausgabesignale (aus Sicht der CU), COND (ALU == 0) ein Eingabesignal.

- Das OP/Ā -Signal c_{14} legt die Quelle für die Mikroadresse des Folgebefehls im CM (Control Memory, Speicher für µBefehle) fest:

  OP/Ā = 0: CMAR ← Adressfeld A (siehe Wilkes-CU)

  OP/Ā = 1: CMAR ← OP /* Adresse des Maschinenbefehls im CM

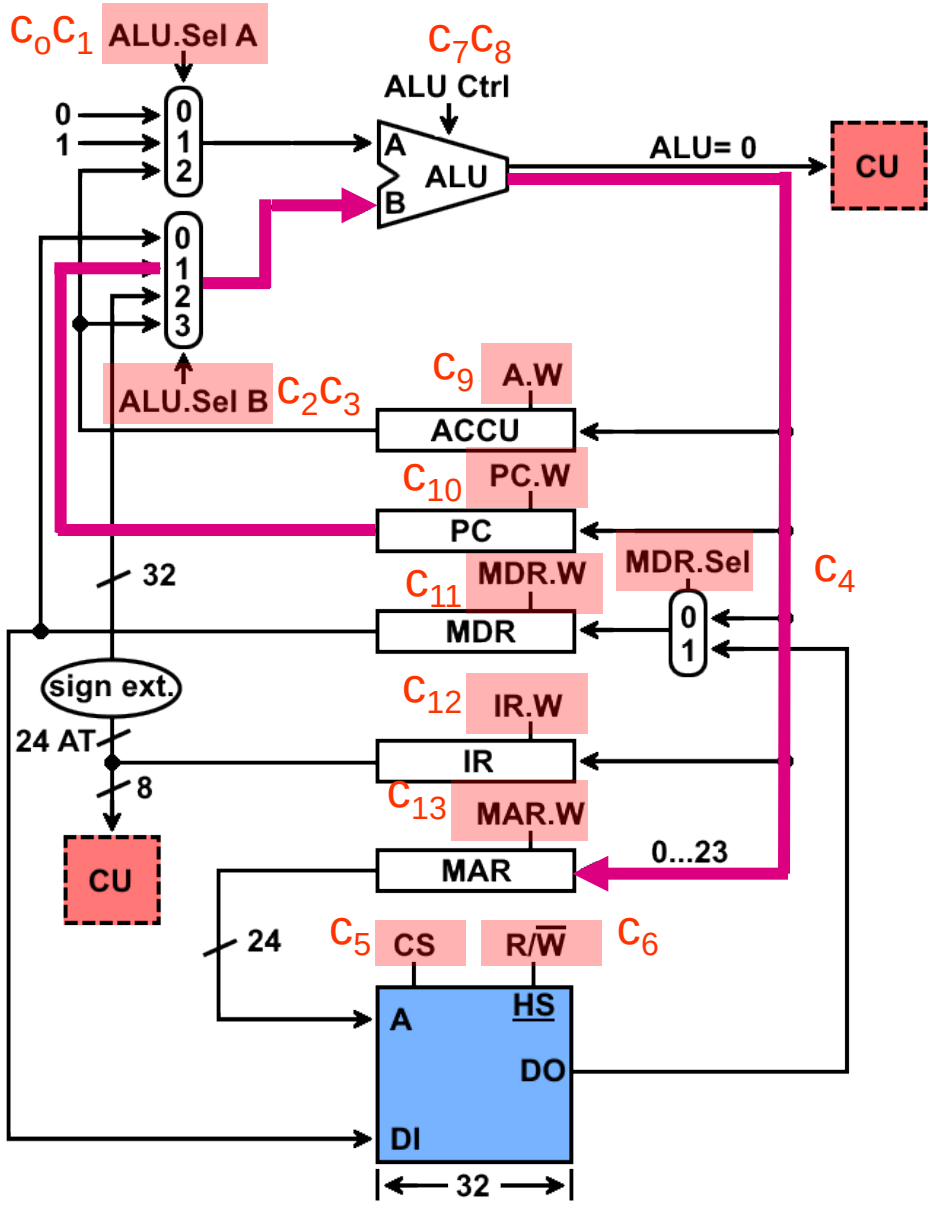


MDR: Memory Data Register
MAR: Memory Address Register
HS.DO: Hauptspeicher.Data out
HS.DI: Hauptspeicher.Data in

Steuersignale für Beispielmaschine (sequenziell)

Label	Adr. CM	ALU Sel. A	ALU Sel.B	MDR Sel	HS CS	HS R/W	ALU Ctrl	ACCU. W	PC.W	MDR.W	IR.W	MAR.W	OP/Ā	COND (ALU==0)	A	Bemerkung
Signalnr.		c0 c1	c2 c3	c4	c5	c6	c7 c8	c9	c10	c11	c12	c13	c14			
IFETCH:	0	x	0 1	x	0	x	TRANS.B	0	0	0	0	1	0	x	1	MAR ← PC;
	1	x	x	1	1	1	x	0	0	1	0	0	0	x	2	MDR ← M[MAR];
	2	x	0 0	x	0	x	TRANS.B	0	0	0	1	0	0	x	3	IR ← MDR;
	3	0 1	0 1	x	0	x	ADD	0	1	0	0	0	0	x	4	PC ← PC + 1;
	4	x	x	x	0	x	x	0	0	0	0	0	1	x	x	GOTO OP;
ADD:	5	x	1 0	x	0	x	TRANS.B	0	0	0	0	1	0	x	6	MAR ← AT;
	6	x	x	1	1	1	x	0	0	1	0	0	0	x	7	MDR ← M[MAR];
	7	0	0 0	x	0	x	ADD	1	0	0	0	0	0	x	8	ACCU←ACCU+MDR;
	8	x	x	x	0	x	x	0	0	0	0	0	0	x	0	GOTO IFETCH;
BZ:	9	1 0	x	x	0	x	TRANS.A	0	0	0	0	0	0 0	1 0	10 0	GOTO JUMP; GOTO IFETCH;
JUMP:	10	x	1 0	x	0	x	TRANS.B	0	1	0	0	0	0	x	11	PC ← AT;
	11	x	x	x	0	x	x	0	0	0	0	0	0	x	0	GOTO IFETCH;

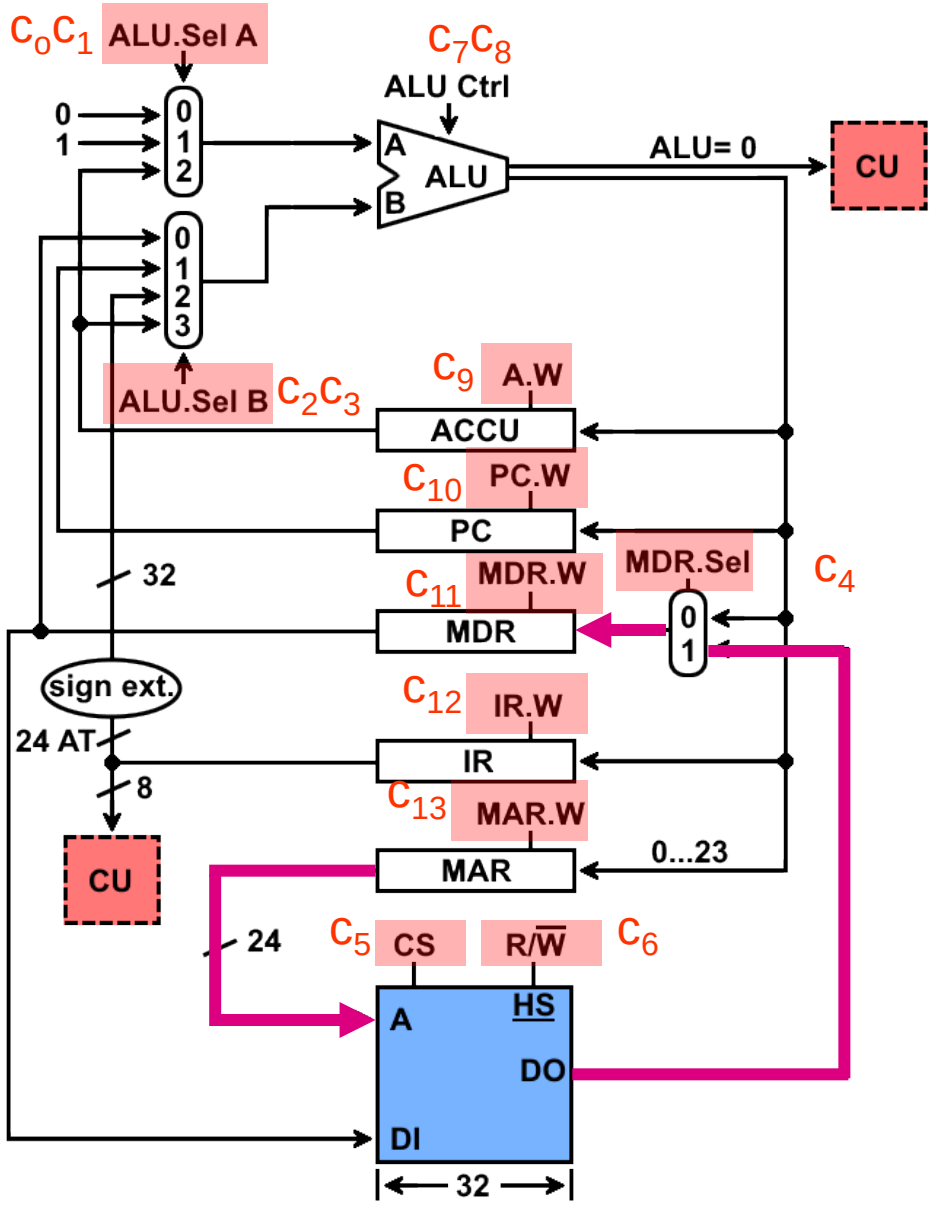
Symbol	ALU Ctrl
ADD	0
TRANS.A	10
TRANS.B	11



MDR: Memory Data Register
MAR: Memory Address Register
HS.DO: Hauptspeicher.Data out
HS.DI: Hauptspeicher.Data in

Beispiel: IFETCH (0) MAR <- PC

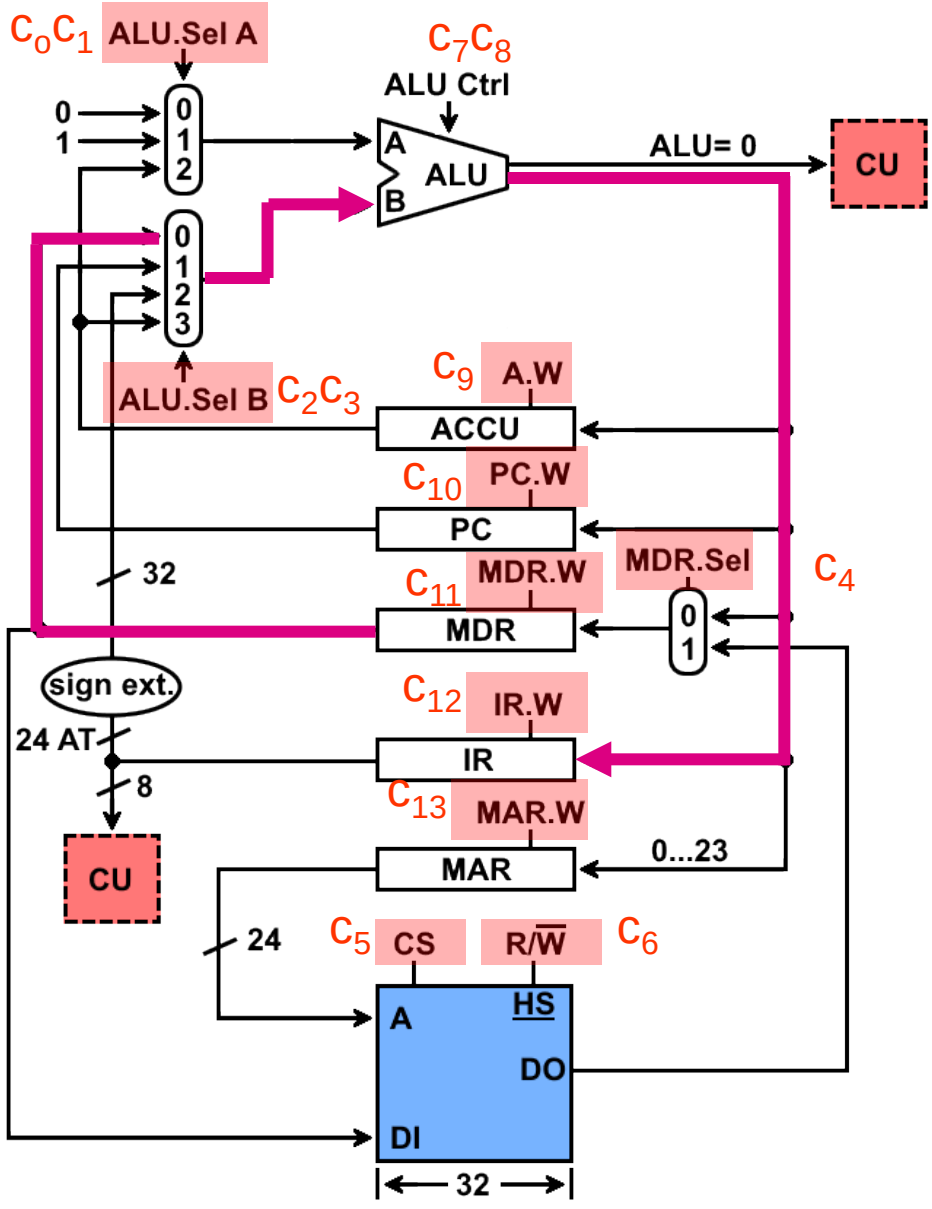
Label	Adr. CM	ALU Sel. A	ALU Sel.B	MDR Sel	HS CS	HS R/W	ALU Ctrl	ACCU. W	PC.W	MDR.W	IR.W	MAR.W	OP/Ā	COND (ALU==0)	A	Bemerkung
Signalnr.		c0 c1	c2 c3	c4	c5	c6	c7 c8	c9	c10	c11	c12	c13	c14			
IFETCH:	0	x	0 1	x	0	x	TRANS.B	0	0	0	0	1	0	x	1	MAR ← PC;
	1	x	x	1	1	1	x	0	0	1	0	0	0	x	2	MDR ← M[MAR];
	2	x	0 0	x	0	x	TRANS.B	0	0	0	1	0	0	x	3	IR ← MDR;
	3	0 1	0 1	x	0	x	ADD	0	1	0	0	0	0	x	4	PC ← PC + 1;
	4	x	x	x	0	x	x	0	0	0	0	0	1	x	x	GOTO OP;
ADD:	5	x	1 0	x	0	x	TRANS.B	0	0	0	0	1	0	x	6	MAR ← AT;
	6	x	x	1	1	1	x	0	0	1	0	0	0	x	7	MDR ← M[MAR];
	7	0	0 0	x	0	x	ADD	1	0	0	0	0	0	x	8	ACCU←ACCU+MDR;
	8	x	x	x	0	x	x	0	0	0	0	0	0	x	0	GOTO IFETCH;
BZ:	9	1 0	x	x	0	x	TRANS.A	0	0	0	0	0	0 0	1 0	10 0	GOTO JUMP; GOTO IFETCH;
JUMP:	10	x	1 0	x	0	x	TRANS.B	0	1	0	0	0	0	x	11	PC ← AT;
	11	x	x	x	0	x	x	0	0	0	0	0	0	x	0	GOTO IFETCH;



MDR: Memory Data Register
MAR: Memory Address Register
HS.DO: Hauptspeicher.Data out
HS.DI: Hauptspeicher.Data in

Beispiel: IFETCH (1) MDR <- M[MAR]

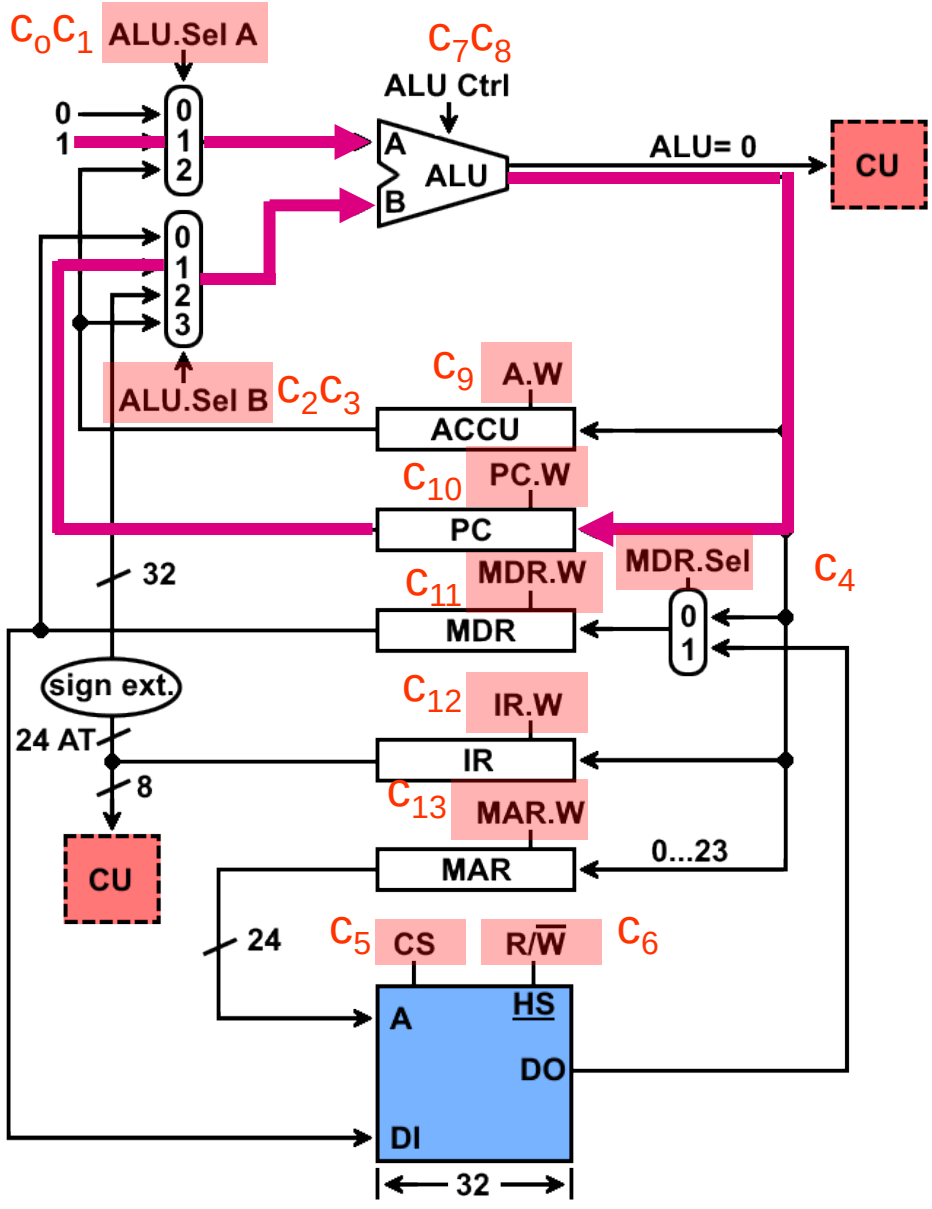
Label	Adr. CM	ALU Sel. A	ALU Sel.B	MDR Sel	HS CS	HS R/W	ALU Ctrl	ACCU. W	PC.W	MDR.W	IR.W	MAR.W	OP/Ā	COND (ALU==0)	A	Bemerkung
Signalnr.		c0 c1	c2 c3	c4	c5	c6	c7 c8	c9	c10	c11	c12	c13	c14			
IFETCH:	0	x	0 1	x	0	x	TRANS.B	0	0	0	0	1	0	x	1	MAR ← PC;
	1	x	x	1	1	1	x	0	0	1	0	0	0	x	2	MDR ← M[MAR];
	2	x	0 0	x	0	x	TRANS.B	0	0	0	1	0	0	x	3	IR ← MDR;
	3	0 1	0 1	x	0	x	ADD	0	1	0	0	0	0	x	4	PC ← PC + 1;
	4	x	x	x	0	x	x	0	0	0	0	0	1	x	x	GOTO OP;
ADD:	5	x	1 0	x	0	x	TRANS.B	0	0	0	0	1	0	x	6	MAR ← AT;
	6	x	x	1	1	1	x	0	0	1	0	0	0	x	7	MDR ← M[MAR];
	7	0	0 0	x	0	x	ADD	1	0	0	0	0	0	x	8	ACCU←ACCU+MDR;
	8	x	x	x	0	x	x	0	0	0	0	0	0	x	0	GOTO IFETCH;
BZ:	9	1 0	x	x	0	x	TRANS.A	0	0	0	0	0	0 0	1 0	10 0	GOTO JUMP; GOTO IFETCH;
JUMP:	10	x	1 0	x	0	x	TRANS.B	0	1	0	0	0	0	x	11	PC ← AT;
	11	x	x	x	0	x	x	0	0	0	0	0	0	x	0	GOTO IFETCH;



MDR: Memory Data Register
MAR: Memory Address Register
HS.DO: Hauptspeicher.Data out
HS.DI: Hauptspeicher.Data in

Beispiel: IFETCH (2) IR <- MDR

Label	Adr. CM	ALU Sel. A	ALU Sel.B	MDR Sel	HS CS	HS R/W	ALU Ctrl	ACCU. W	PC.W	MDR.W	IR.W	MAR.W	OP/Ā	COND (ALU==0)	A	Bemerkung
Signalnr.		c0 c1	c2 c3	c4	c5	c6	c7 c8	c9	c10	c11	c12	c13	c14			
IFETCH:	0	x	0 1	x	0	x	TRANS.B	0	0	0	0	1	0	x	1	MAR ← PC;
	1	x	x	1	1	1	x	0	0	1	0	0	0	x	2	MDR ← M[MAR];
	2	x	0 0	x	0	x	TRANS.B	0	0	0	1	0	0	x	3	IR ← MDR;
	3	0 1	0 1	x	0	x	ADD	0	1	0	0	0	0	x	4	PC ← PC + 1;
	4	x	x	x	0	x	x	0	0	0	0	0	1	x	x	GOTO OP;
ADD:	5	x	1 0	x	0	x	TRANS.B	0	0	0	0	1	0	x	6	MAR ← AT;
	6	x	x	1	1	1	x	0	0	1	0	0	0	x	7	MDR ← M[MAR];
	7	0	0 0	x	0	x	ADD	1	0	0	0	0	0	x	8	ACCU←ACCU+MDR;
	8	x	x	x	0	x	x	0	0	0	0	0	0	x	0	GOTO IFETCH;
BZ:	9	1 0	x	x	0	x	TRANS.A	0	0	0	0	0	0 0	1 0	10 0	GOTO JUMP; GOTO IFETCH;
JUMP:	10	x	1 0	x	0	x	TRANS.B	0	1	0	0	0	0	x	11	PC ← AT;
	11	x	x	x	0	x	x	0	0	0	0	0	0	x	0	GOTO IFETCH;



MDR: Memory Data Register
MAR: Memory Address Register
HS.DO: Hauptspeicher.Data out
HS.DI: Hauptspeicher.Data in

Beispiel: IFETCH (3) PC <- PC+1

Label	Adr. CM	ALU Sel. A	ALU Sel.B	MDR Sel	HS CS	HS R/W	ALU Ctrl	ACCU. W	PC.W	MDR.W	IR.W	MAR.W	OP/Ā	COND (ALU==0)	A	Bemerkung
Signalnr.		c0 c1	c2 c3	c4	c5	c6	c7 c8	c9	c10	c11	c12	c13	c14			
IFETCH:	0	x	0 1	x	0	x	TRANS.B	0	0	0	0	1	0	x	1	MAR ← PC;
	1	x	x	1	1	1	x	0	0	1	0	0	0	x	2	MDR ← M[MAR];
	2	x	0 0	x	0	x	TRANS.B	0	0	0	1	0	0	x	3	IR ← MDR;
	3	0 1	0 1	x	0	x	ADD	0	1	0	0	0	0	x	4	PC ← PC + 1;
	4	x	x	x	0	x	x	0	0	0	0	0	1	x	x	GOTO OP;
ADD:	5	x	1 0	x	0	x	TRANS.B	0	0	0	0	1	0	x	6	MAR ← AT;
	6	x	x	1	1	1	x	0	0	1	0	0	0	x	7	MDR ← M[MAR];
	7	0	0 0	x	0	x	ADD	1	0	0	0	0	0	x	8	ACCU←ACCU+MDR;
	8	x	x	x	0	x	x	0	0	0	0	0	0	x	0	GOTO IFETCH;
BZ:	9	1 0	x	x	0	x	TRANS.A	0	0	0	0	0	0 0	1 0	10 0	GOTO JUMP; GOTO IFETCH;
JUMP:	10	x	1 0	x	0	x	TRANS.B	0	1	0	0	0	0	x	11	PC ← AT;
	11	x	x	x	0	x	x	0	0	0	0	0	0	x	0	GOTO IFETCH;

- Wir bezeichnen jede Zeile der Steuersignaltabelle als Mikrobefehl.
  Er setzt sich aus einer oder mehreren Mikrooperationen (= RT-Operationen) zusammen.

- Mikrooperationen können parallel ablaufen, Mikrobefehle nacheinander.

- Aus der für einen bestimmten Maschinenbefehl (Instruction) nötigen Anzahl von Mikrobefehlen bestimmt sich das CPI (Cycles per Instruction) für diesen Befehl.

IFETCH:	0	x	0	1	x	0	x	TRANS.B	0	0	0	0	1	0	x	1	MAR ← PC;
	1	x	x		1	1	1	x	0	0	1	0	0	0	x	2	MDR ← M[MAR];
	2	x	0	0	x	0	x	TRANS.B	0	0	0	1	0	0	x	3	IR ← MDR;
	3	0	1	0	1	x	0	x	ADD	0	1	0	0	0	0	x	4	PC ← PC + 1;

Mikrobefehl

Mikrooperationen

- In welcher Reihenfolge werden die Befehle für den IFETCH (Laden eines Befehls) durchgeführt?

  - GOTO OP
  - IR <- MDR
  - MAR <- PC
  - PC <- PC+1
  - MDR <- M[MAR]

- Grundsätzlich lassen sich RT-Operationen gleichzeitig (parallel) ausführen, sofern sie keine gemeinsamen Ressourcen benutzen.

- Im Beispiel der Minimax ist das kaum der Fall, da die ALU für die TRANSFER-Operationen verwendet wird. Trotzdem lassen sich die Operationen GOTO IFETCH, GOTO OP und PC ← PC + 1 parallel durchführen.

- Bei komplexeren Prozessoren sind mehrere ALUs mit Spezialisierung vorhanden (z.B. arithmetische Operationen, Adressrechnung, Befehlszählerinkrement, logische Operationen)

Steuersignale für Beispielmaschine (parallel)

Label	Adr. CM	ALU Sel. A	ALU Sel.B	MDR Sel	HS CS	HS R/W	ALU Ctrl	ACCU. W	PC.W	MDR.W	IR.W	MAR.W	OP/Ā	COND ALU==0	A	Bemerkung
Signalnr.		c0 c1	c2 c3	c4	c5	c6	c7 c8	c9	c10	c11	c12	c13	c14			
IFETCH:	0	x	0 1	x	0	x	TRANS.B	0	0	0	0	1	0	x	1	MAR ← PC;
	1	0 1	0 1	1	1	1	ADD	0	1	1	0	0	0	x	2	MDR ← M[MAR]; PC ← PC +1;
	2	x	0 0	x	0	x	TRANS.B	0	0	0	1	0	1	x	x	IR ← MDR; GOTO OP;
ADD:	3	x	1 0	x	0	x	TRANS.B	0	0	0	0	1	0	x	4	MAR ← AT;
	4	x	x	1	1	1	x	0	0	1	0	0	0	x	5	MDR ← M[MAR];
	5	1 0	0 0	x	0	x	ADD	1	0	0	0	0	0	x	0	ACCU←ACCU+MDR; GOTO IFETCH;
BZ:	6	1 0	x	x	0	x	TRANS.A	0	0	0	0	0	0 0	1 0	7 0	GOTO JUMP; GOTO IFETCH;
JUMP:	7	x	1 0	x	0	x	Trans.B	0	1	0	0	0	0	x	0	PC ← AT : GOTO IFETCH;

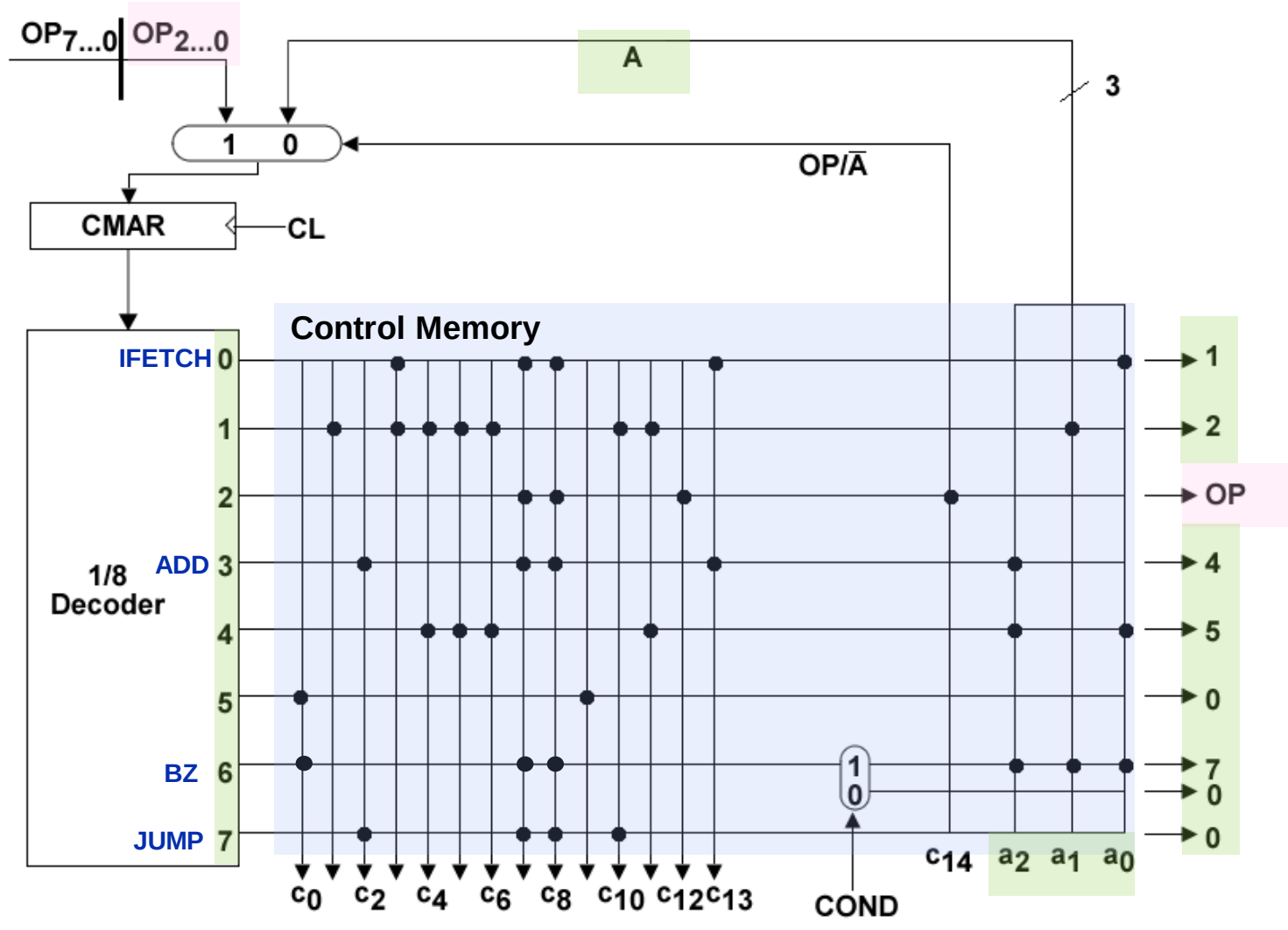
Reduktion des CM von 12 Zeilen (sequentiell) auf 8 Zeilen (parallel)

CMAR <- OP:
OP bestimmt Anfangsadresse für die Operation in CM

CMAR <- A:
A bestimmt Folge-adresse für die Abarbeitung des µBefehls

Der Opcode erlaubt 256 Befehle. Hier werden nur 3 bit benutzt.

Die hier gezeigten Steuersignale entsprechen der parallelen Version.



CU für die Beispielmaschine MINIMAX

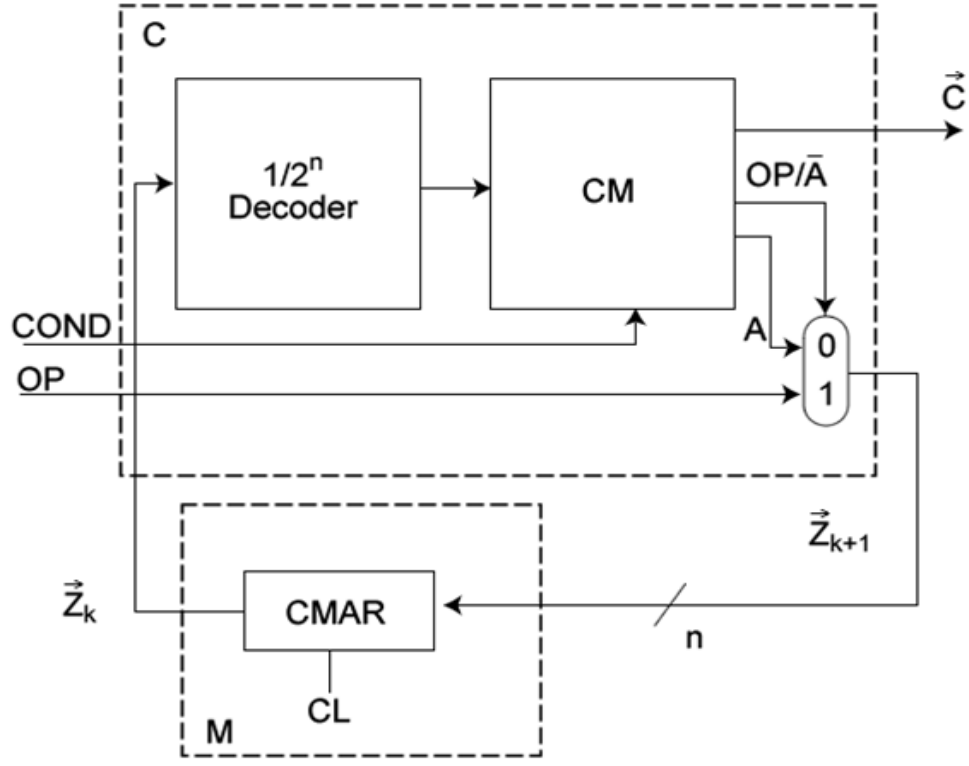
RA - T11 - 6

- Das Control Memory Address Register CMAR (im Beispiel 3 bit weit) zeigt auf eine der 8 Zeilen des Control Memory CM.
- Die aktivierte Zeile setzt die mit einem Punkt versehenen Steuersignale c_i auf 1.
- Ist OP/$\bar{A}$ = 0, dann wird die in (a_2 a_1 a_0) codierte Folgeadresse A in CMAR geladen.
- Für OP/$\bar{A}$ = 1 wird der Opcode OP inCMAR geladen.
  - ADD: Opcode 3, BZ: Opcode 6, JUMP: Opcode 7
- Über COND kann eine von 2 alternativen Folgeadressen gewählt werden.

- Die Wilkes-CU lässt sich als spezielle Form eines endlichen Automaten auffassen:

$$\vec{C}_k = f(COND, OP, \vec{Z}_k)$$

- Die Steuersignaltabelle (= Mikrobefehlstabelle) kann in einen endlichen Automaten umgeformt werden. Dazu wird jedem Mikrobefehl ein Zustand zugeordnet.
- Durch die Folgeadressen werden die Zustandsübergänge gesteuert.
- Für RISC-Prozessoren werden üblicherweise endliche Automaten als Steuerung verwendet.



RA - T11 - 7

Umformung der Wilkes-CU in einen Endlichen Automaten (2^n Zustände)

- Die Wilkes-CU lässt sich als spezielle Form eines endlichen Automaten auffassen. Handelt es sich dabei um:
  - Einen Moore Automaten

    oder

  - Einen Mealy Automaten

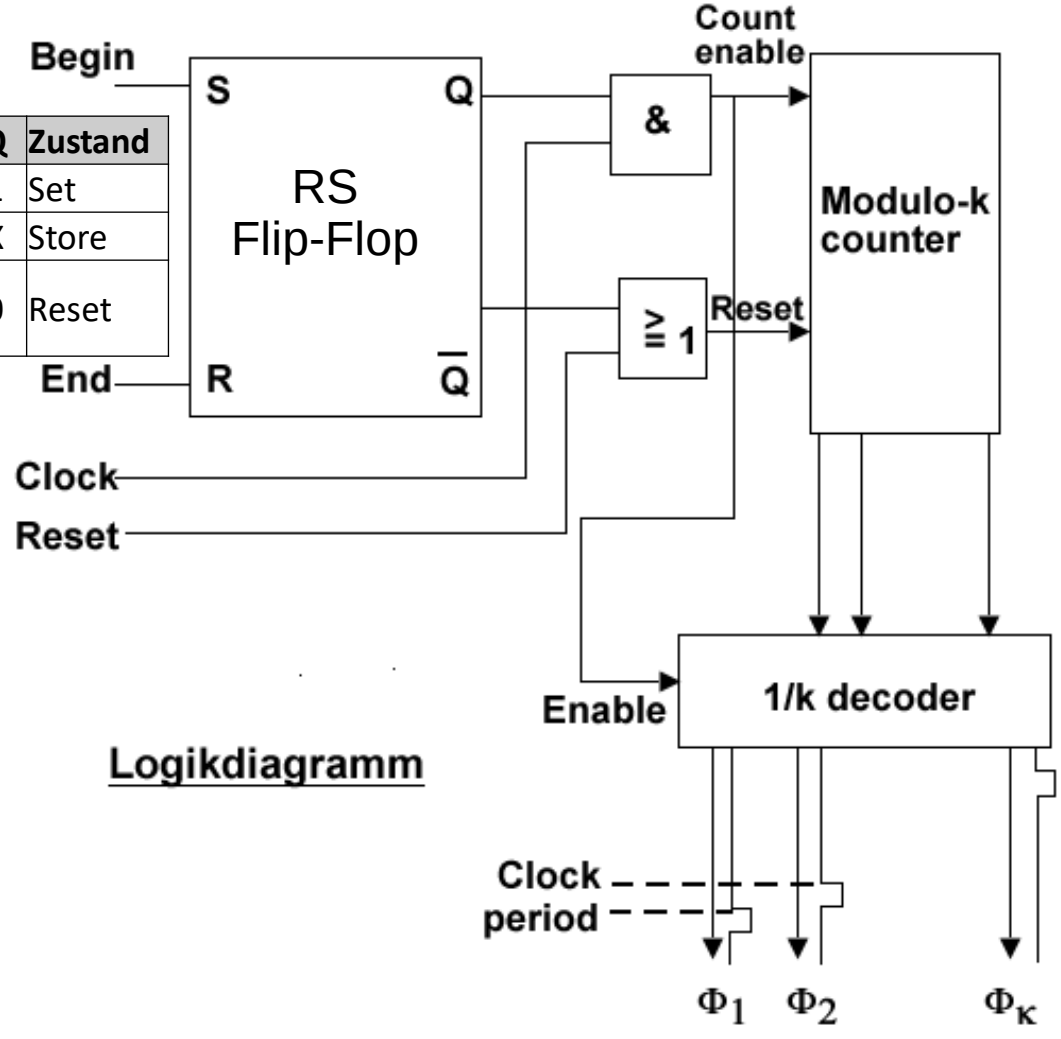
- Die Trefferquote liegt bei 50%, mit Überlegung kann man sie auf 100% steigern!

- Mit Hilfe eines Modulo-k-Sequenzzählers kann eine feste Abfolge von Einzelimpulsen Φ_i erzeugt werden.

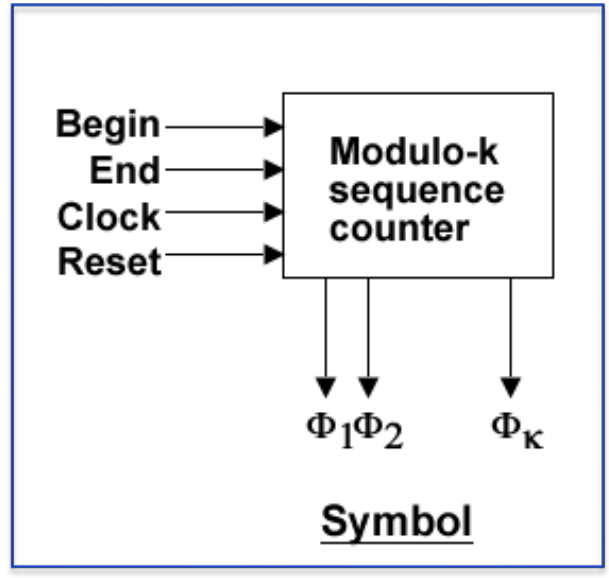
- Diese Art von Steuerung korrespondiert mit Abläufen, die nach einer festen Schrittzahl von vorne beginnen, z.B. der Abarbeitung eines Befehlszyklus.

- Durch eine Kombinatorik werden aus den Phasensignalen Φ_i die Steuersignale c_{out} erzeugt.

S	R	Q	Zustand
1	0	1	Set
0	0	X	Store
0	1	0	Reset

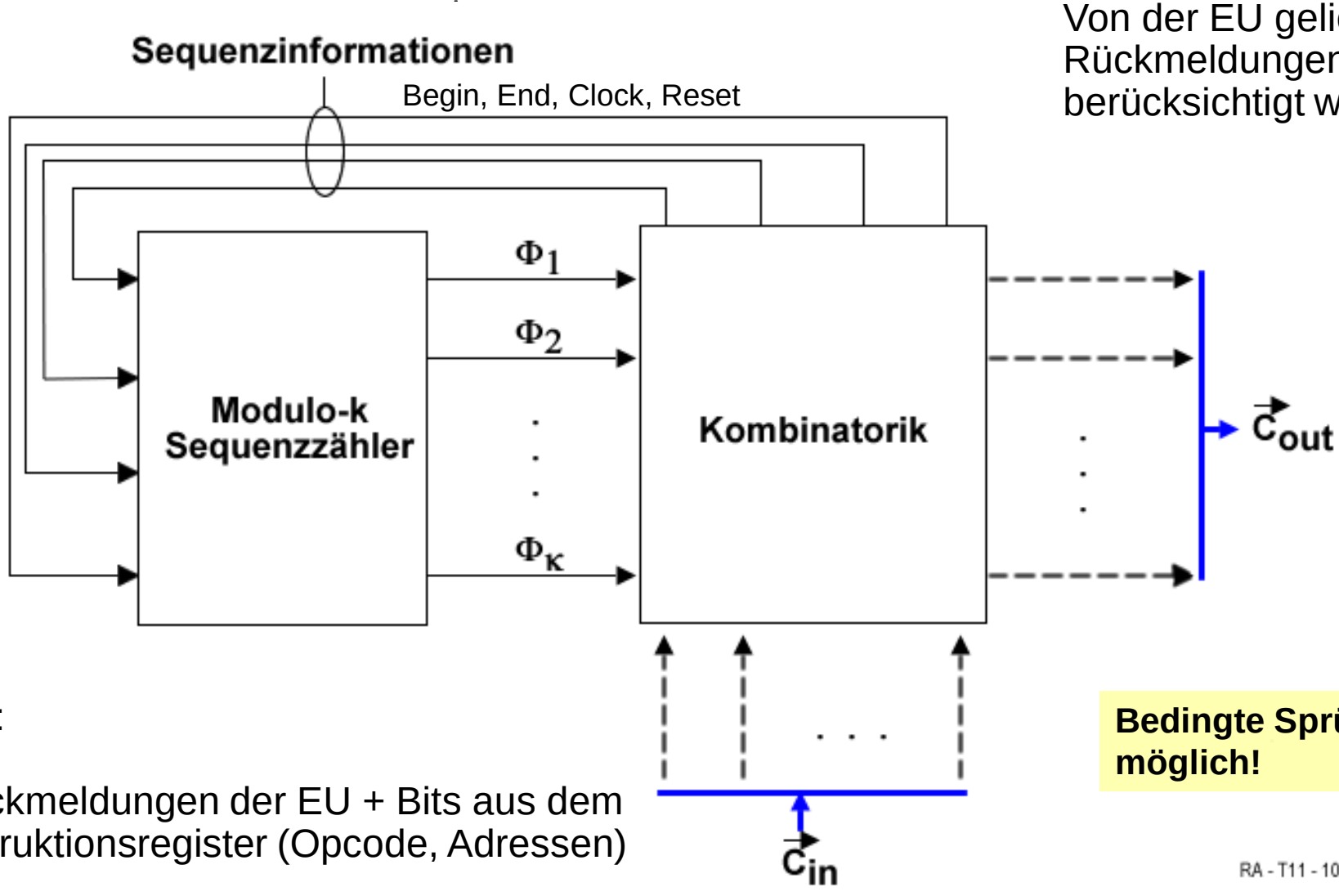


Logikdiagramm



Bedingte Sprünge so nicht möglich, Feedback fehlt.

RA - T11 - 9



Von der EU gelieferte Rückmeldungen c_{in} können berücksichtigt werden.

$\vec{c}_{in}$:

Rückmeldungen der EU + Bits aus dem Instruktionsregister (Opcode, Adressen)

Bedingte Sprünge möglich!

RA - T11 - 10

- Festverdrahtete Steuerungen (z.B. Wilkes, Sequenzzähler) sind schnell und mit relativ geringem HW-Aufwand zu realisieren. Dies gilt vor allem für kleinere Befehlssätze, die typisch sind für RISC-Prozessoren.

- Sie haben aber auch Nachteile:
  - Sie sind unflexibel. Fehler führen möglicherweise zu großen HW-Änderungen.
  - Das CM enthält für jeden bedingten Sprungbefehl einen MUX. Es lässt sich also nicht als Standard-ROM oder -RAM realisieren.

- Alternative: Mikroprogrammierung

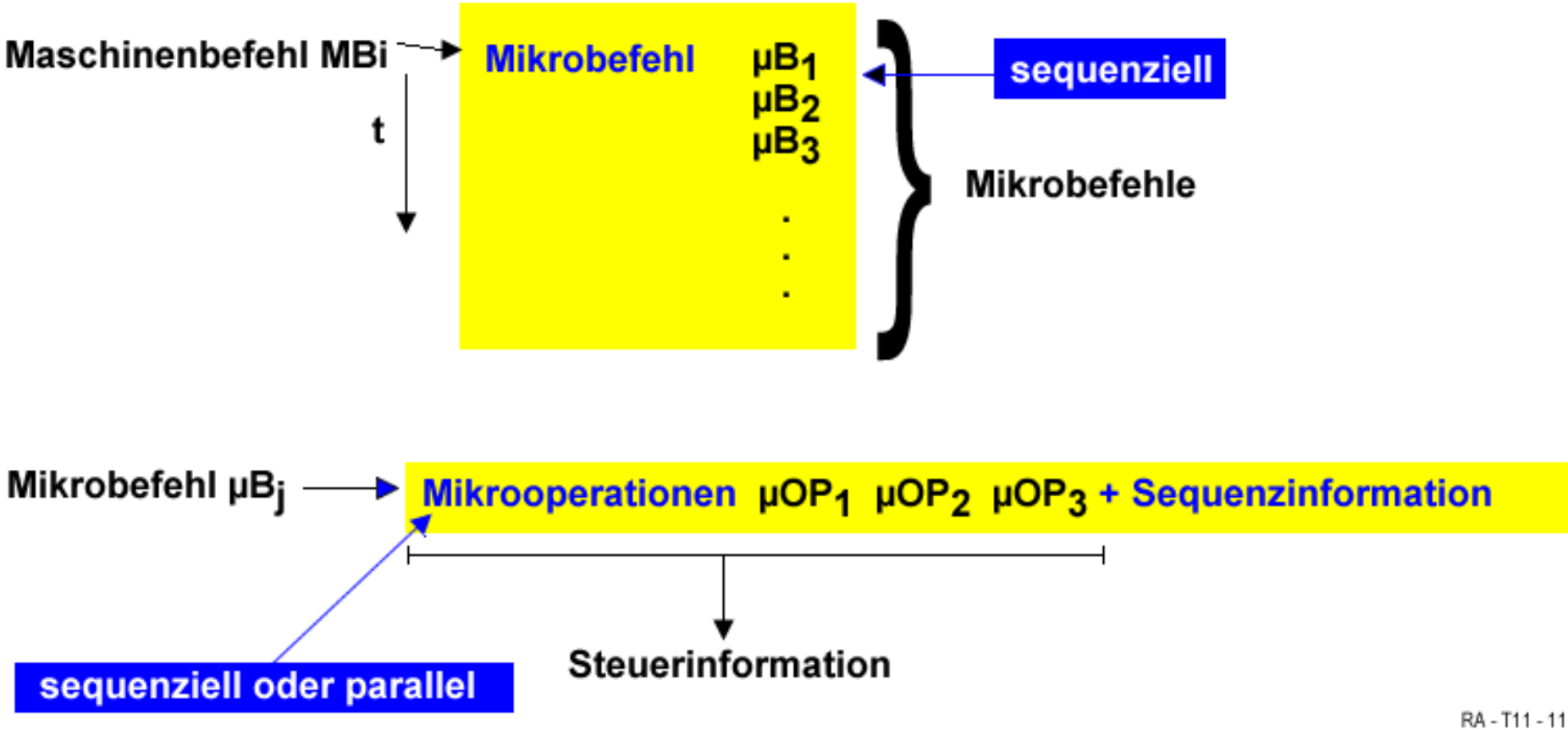
- µprogrammierter Prozessor:
  µProgramm durch Hersteller erzeugt, steht im ROM

- µprogrammierbarer Prozessor:
  µProgramm im RAM, Änderung durch Anwender möglich

- dynamisch µprogrammierbarer Prozessor:
  Inhalt des Control Memory kann zur Laufzeit verändert werden
  (writeable control store WCS).

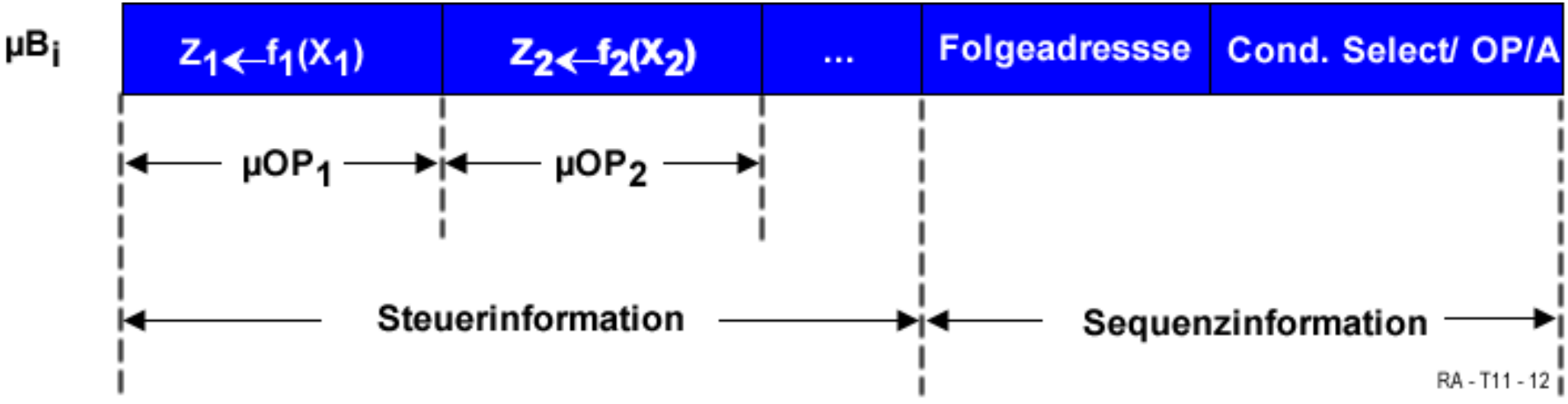
- (Firmware-)Emulator: Menge der Mikroprogramme, welche einen bestimmten Befehlssatz implementieren

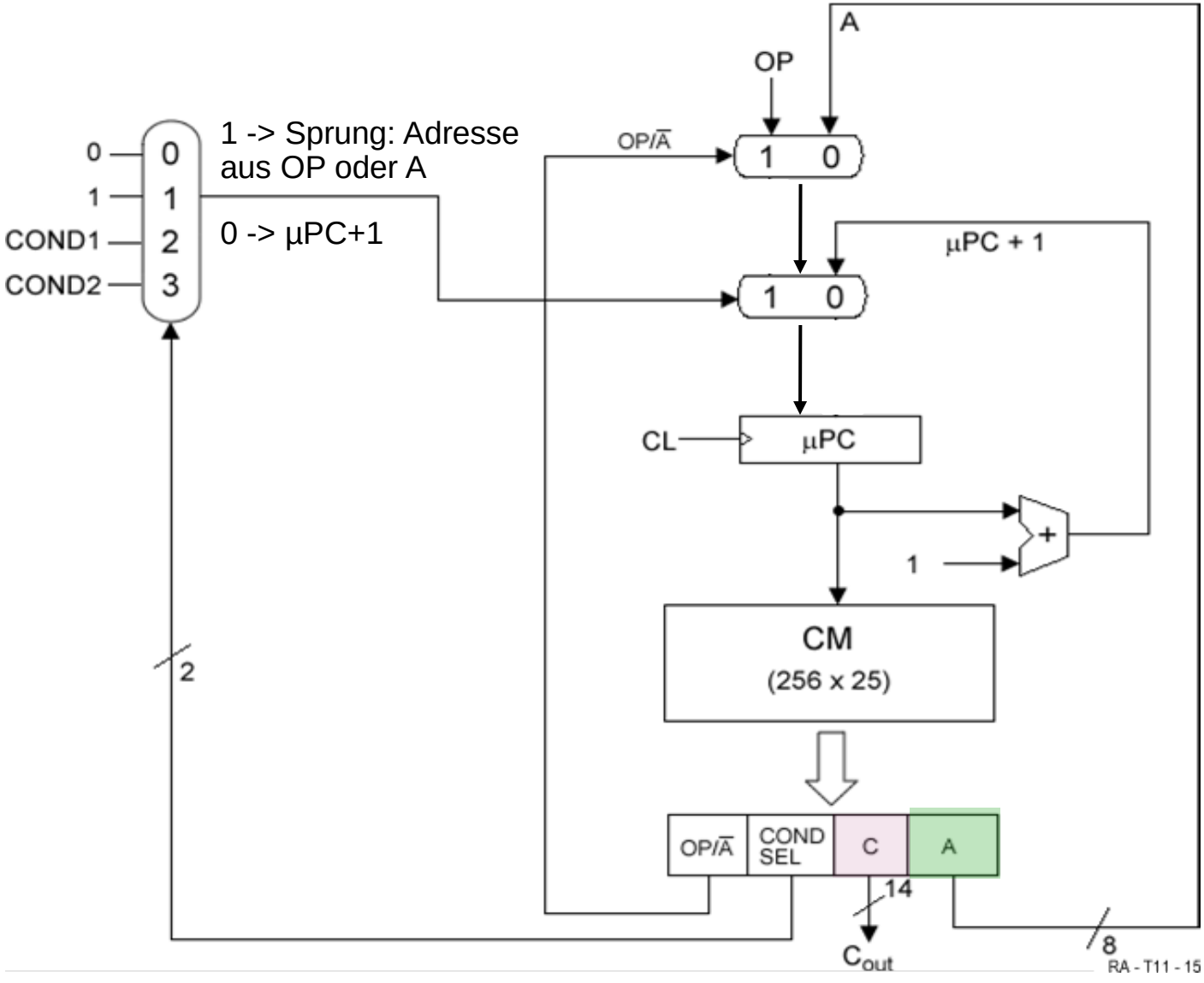
- Firmware: Inhalt von CM



RA - T11 - 11

- Die Sequenzinformation besteht aus der Folgeadresse A sowie einem Code, der für bedingte Sprünge benutzt wird (Condition Select, OP/Ā etc.).
- Steuerinformationen = Folge von Mikrooperationen
- Mikrooperation = RT-Befehl





Mikroprogramm steht im Control Memory und kann bei Bedarf geändert werden

- Eine mikroprogrammierte Steuerung ist eine Abwandlung der Wilkes-CU mit folgenden Änderungen:
  - Die Bedingungsauswahl ist aus dem CM herausverlagert.
  - Das CMAR kann inkrementiert werden. Es heißt jetzt Micro Program Counter (µPC).
  - Das Mikrobefehlswort wird erweitert. Es enthält jetzt folgende Felder:

    C: Steuersignale C_{out}

    A: Folgeadresse für Sprung im CM

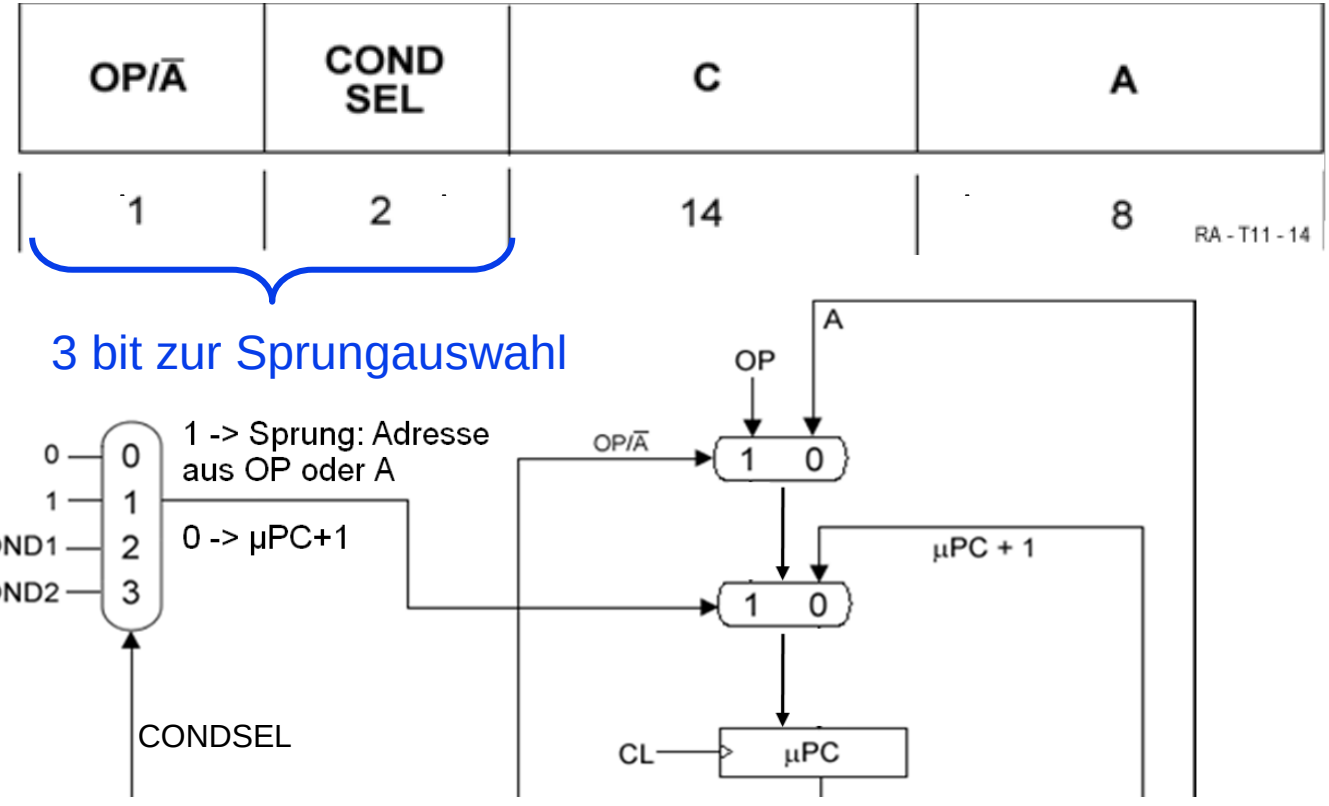
    CONDSEL: "Condition Select" wählt aus, welche externe Statusvariable (je 1 Bit aus SR) für die Sprungentscheidung bei bedingten Sprüngen verwendet werden soll.

    OP/Ā : Das 1 bit-Signal wählt aus, ob im Fall eines Sprungs die Folgeadresse intern (A) oder extern (OP) bereitgestellt werden soll.

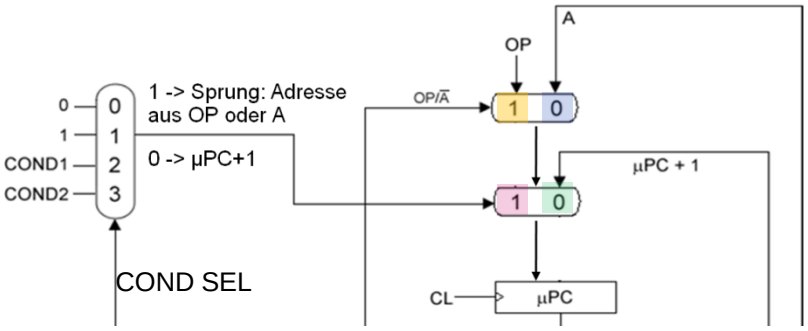
μB_i	µOPCODE	µOPs oder Folgeadresse	Cond. Select

Mikrobefehls-format (2)

- µOPCODE (evtl.): Im Fall von Nicht-Sprungbefehlen wird das A-Feld nicht benötigt. Sprungbefehle brauchen kein C-Feld (Steuerinformationen). Durch den OPCODE kann also z.B. zwischen 2 Mikrobefehlsformaten ausgewählt werden (Platzersparnis im CM).



Mikrobefehl: Codierung der Sprungauswahl



	Sprungauswahl			
Sprungaus-wahl$_{10}$	OP/Ā	COND SEL	SPRUNG	Bemerkung
0	0	0 0	0	µPC ← µPC + 1;
1	0	0 1	1	µPC ← A;
2	0	1 0	COND1	if COND1: µPC ← A else: µPC ← µPC + 1;
3	0	1 1	COND2	if COND2: µPC ← A else: µPC ← µPC + 1;
4	1	0 0	0	µPC ← µPC + 1;
5	1	0 1	1	µPC ← OP;
6	1	1 0	COND1	if COND1: µPC ← OP else: µPC ← µPC + 1;
7	1	1 1	COND2	if COND2: µPC ← OP else: µPC ← µPC + 1;

linear

Sprung im µProgramm

Bedingter Sprung im µProgramm

linear

Sprung zu neuer OP

Bedingter Sprung zu Neuer OP

- Die folgende Tabelle zeigt das MINIMAX-Mikroprogramm.
- Der Steuersignalteil ist unverändert von der Wilkes-CU übernommen. Nur das Signal GOTO OP entfällt.
- Ergänzt sind die Felder OP/Ā, CONDSEL und A.
- Es ist angenommen, dass das Statussignal ALU == 0 mit COND1 verbunden ist.
- Da sich hier nur eine Folgeadresse A codieren lässt, ist das Mikroprogramm für BZ um einen Schritt länger (Zeile mit Nr. 7 kommt dazu)

Label	Adr. CM_{10}	OP/A	COND SEL	ALU Sel.A	ALU Sel.B	MDR Sel	HS CS	HS R/W	ALU Ctrl	ACC U.W	PC. W	MDR. W	IR.W	MAR. W	A_{10}	Bemerkung
Signalnr.				c0 c1	c2 c3	c4	c5	c6	c7 c8	c9	c10	c11	c12	c13		
IFETCH:	0	x	0 0	x x	0 1	x	0	x	1 1	0	0	0	0	1	x	MAR ← PC;
	1	x	0 0	0 1	0 1	1	1	1	0 0	0	1	1	0	0	x	MDR ← M[MAR]; PC ← PC + 1;
	2	1	0 1	x x	0 0	x	0	x	1 1	0	0	0	1	0	OP	IR ← MDR: GOTO OP;
ADD:	3	x	0 0	x x	1 0	x	0	x	1 1	0	0	0	0	1	x	MAR ← AT;
	4	x	0 0	x x	x x	1	1	1	x x	0	0	1	0	0	x	MDR ← M[MAR];
	5	0	0 1	1 0	0 0	x	0	x	0 0	1	0	0	0	0	0	ACCU ← ACCU + MDR; GOTO IFETCH;
BZ:	6	0	1 0	1 0	x x	x	0	x	1 0	0	0	0	0	0	8	If COND1 : GOTO JUMP;
	7	0	0 1	x x	x x	x	0	x	x x	0	0	0	0	0	0	GOTO IFETCH;
JUMP:	8	0	0 1	x x	1 0	x	0	x	1 1	0	1	0	0	0	0	PC ← AT : GOTO IFETCH;

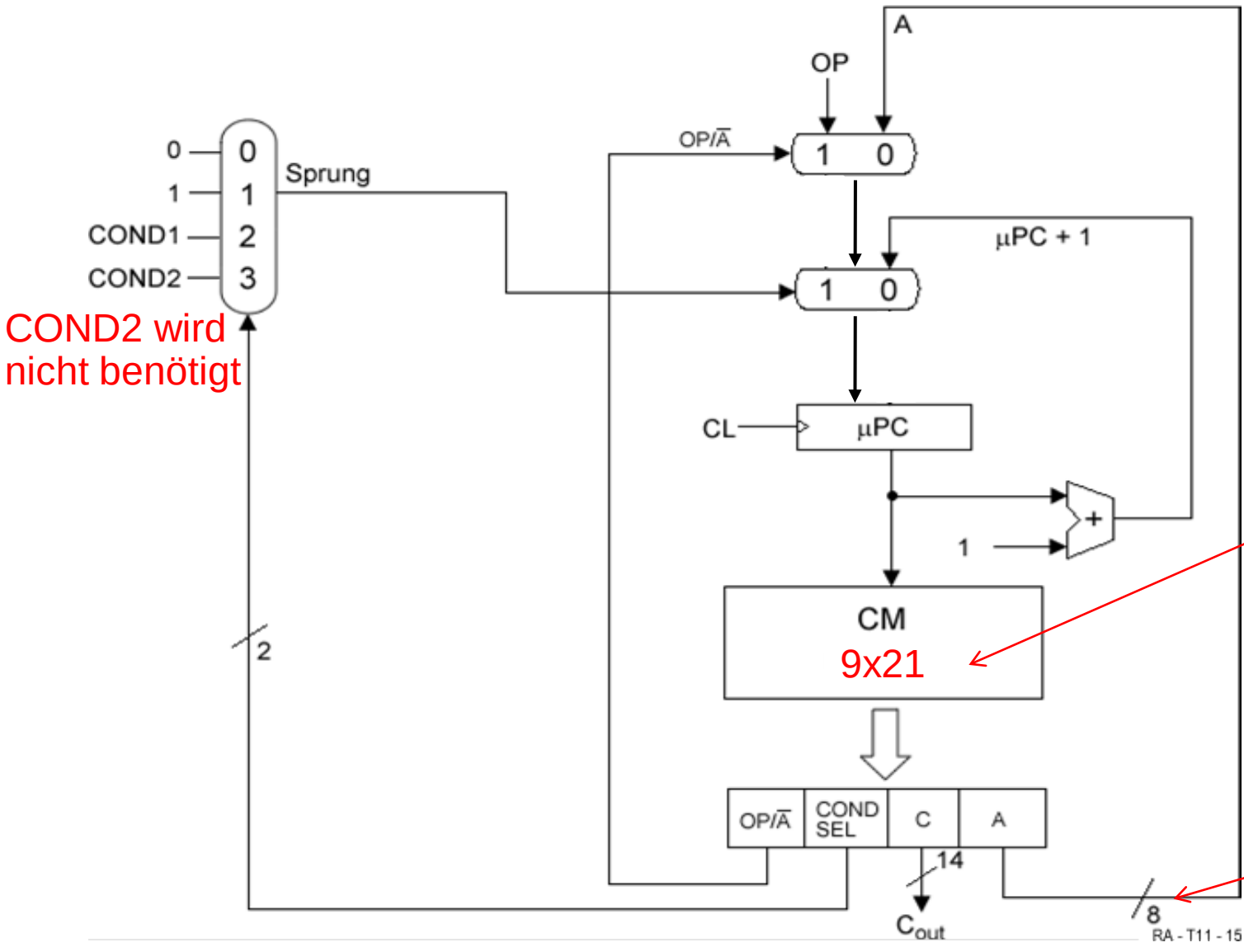
Steuersignale „Nutzlast“:
3Bit OP/A und Cond + 14 Bit Control + 8 Bit Adressen

Control Memory CM

rot: GOTO

grün: linear μPC + 1

Blau: Sequenzinformation



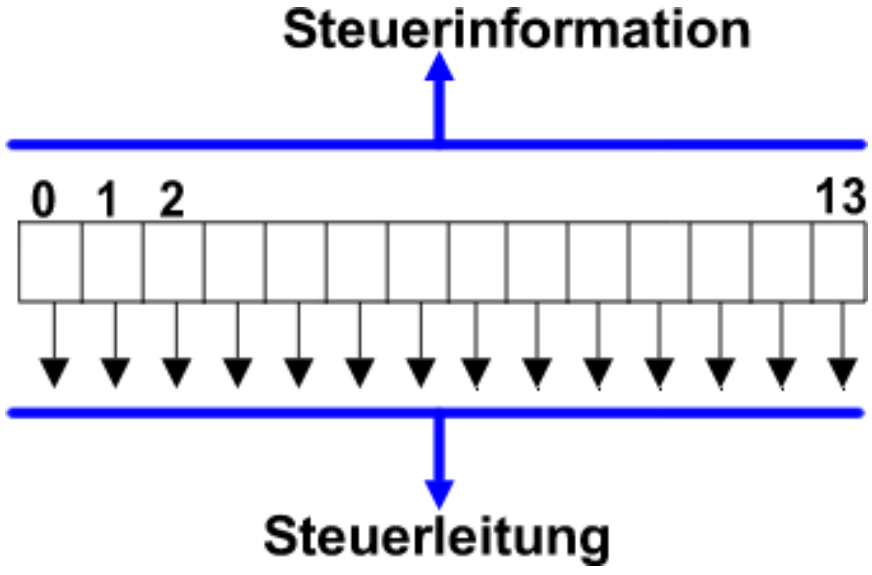
COND2 wird nicht benötigt

Zeilen 0-2: IFETCH
Zeilen 3-5: ADD
Zeilen 6-7: BZ
Zeile 8: JUMP

Nur 4 Bits werden für 9 Zeilen benötigt

- Bisher hatten wir angenommen, dass jeder Steuerleitung ein Bit im Feld Steuerinformation zugeordnet ist. Diese Technik heißt horizontale Mikroprogrammierung.

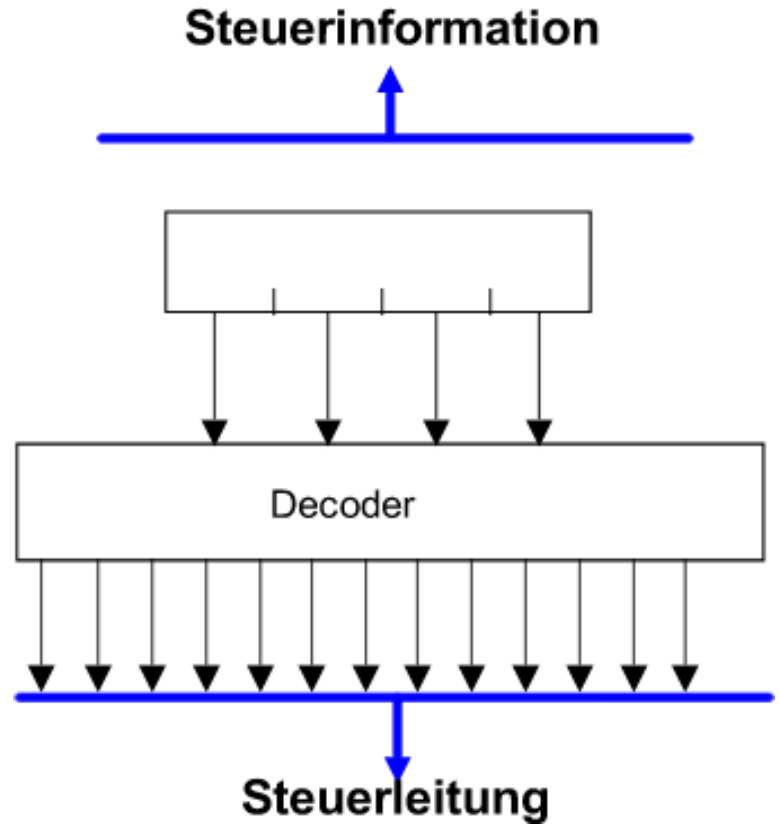
- In einem uncodierten (horizontalen) Steuerinformationsfeld können beliebig viele Stellen auf 1 gesetzt sein. Dies entspricht der gleichzeitigen (parallelen) Ausführung der entsprechenden Mikrooperationen.



keine Codierung

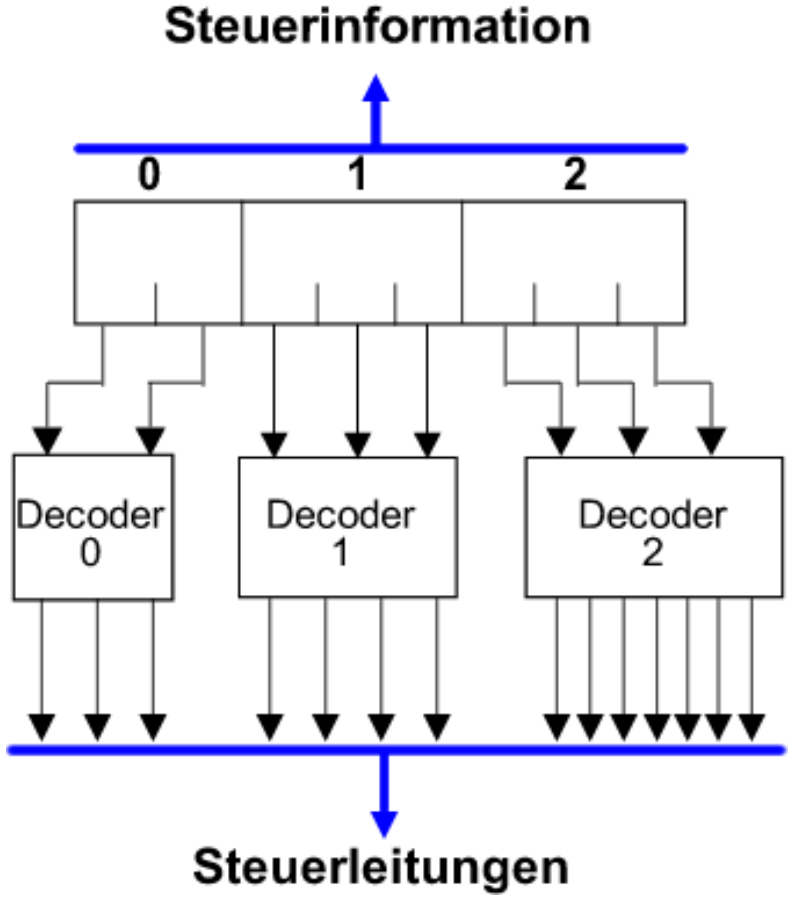
- Codiert man die Steuerinformation, so erreicht man eine Platzersparnis im CM, allerdings erkauft durch einen nun zusätzlich nötigen Decodierungsschritt (vertikale Mikroprogrammierung).

- Am Ausgang eines Decoders ist genau eine Leitung aktiv. Die vertikale Mikroprogrammierung begrenzt also die Parallelität auf Hardwareebene.



vollständige Codierung

- Ein Kompromiss ist die teilweise Codierung (quasi-horizontale oder diagonale Mikroprogrammierung).



teilweise Codierung

- horizontale M.P.:
  schnell, teuer, parallel, >100 bit Steuerinformation (sehr große Wortlänge für Mikrobefehle: 1 bit pro µOP)

- quasi-horizontale M.P.:
  Kompromiss zwischen horizontaler und vertikaler Mikroprogrammierung.
  Volle Nutzung der Parallelität möglich durch Gruppenbildung.
  Nachteil: Decodieraufwand

- vertikale M.P.:
  kleine Wortlänge (< 30 bit)
  Nachteile:
  Beschränkung der Parallelität, große Mikroprogrammlänge, µProgramm-Adressen lang